

全国计算机技术与软件专业技术资格（水平）考试参考用书

系统分析师技术指南

全国计算机专业技术资格考试办公室推荐

张友生 王勇 主编　　希赛IT教育研发中心 组编

根据2009版大纲编写

清华大学出版社

全国计算机与软件专业技术资格（水平）考试参考用书

系统分析师技术指南

张友生　王勇　主编

希赛 IT 教育研发中心　组编

清 华 大 学 出 版 社
北　京

内 容 简 介

本书对当前比较前沿而又成熟的技术和方法进行了讨论，包括软件过程改进、J2EE 与.NET 平台、中间件及相关技术、应用服务器、Web Service、数据仓库与数据挖掘、操作数据存储、异构数据库的集成、企业应用集成、XML、软件架构、设计模式、SOA、RIA、UML、UP、SOAP、PDM/PLM、AOP、P2P、工作流、软件产品线、敏捷方法、网格计算与普适计算、云计算与 SaaS、多核技术、片上系统等。这些技术和方法是任何一位合格的系统分析师必须具备的知识，也是系统分析师考试必考的知识点。阅读本书，犹如进入 IT 新技术和新方法的殿堂。

本书由希赛 IT 教育研发中心组编，作为计算机技术与软件专业技术资格（水平）考试参考用书，同时也可作为系统分析师日常工作的参考手册，作为软件设计师、数据库系统工程师、网络工程师进一步深造和发展的必读书籍，也是计算机专业教师的教学和工作参考书。

图书在版编目(CIP)数据

系统分析师技术指南/张友生，王勇主编. —北京：清华大学出版社，2009.8
（全国计算机技术与软件专业技术资格（水平）考试参考用书）
ISBN 978-7-302-20647-7

Ⅰ. 系…　Ⅱ. ①张…②王…　Ⅲ. 软件工程-系统分析-工程技术人员-资格考核-自学参考资料　Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字（2009）第 123548 号

责任编辑：柴文强　赵晓宁
责任校对：徐俊伟
责任印制：
出版发行：清华大学出版社　　**地　　址：**北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185×230　**印　张：**29.25　**防伪页：**1　**字　数：**674 千字
版　　次：2009 年 8 月第 1 版　**印　次：**2009 年 8 月第 1 次印刷
印　　数：
定　　价：　　元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　产品编号：033362-01

前　　言

系统分析是 IT 组织开发优秀的应用系统的重要工作，需要拥有扎实的理论知识和丰富的实际经验的人员来完成。随着应用系统规模越来越大，复杂程度越来越高，系统分析师在系统开发的过程中，发挥着越来越重要的作用。

全国计算机技术与软件专业资格（水平）考试作为培养和选拔计算机技术和软件专业人才的一个考试，其难度较大。主要原因是考试范围十分广泛，牵涉到计算机专业的每门课程，还要加上数学，外语，系统工程，信息化和知识产权等知识，且注重考查新技术和新方法的应用。考试不但注重广度，而且还有一定的深度。特别是高级资格考试，不但要求考生具有扎实的理论知识，还要具有丰富的实践经验。

1. 目的

众所周知，IT 技术日新月异，作为一名合格的系统分析师，必须善于学习，及时了解和掌握新技术与新方法。然而，考生又不可能有足够的时间和精力去详细学习和实践当前的每一种 IT 新技术和新方法，因此，就需要一本书来对当前比较流行而又成熟的技术和方法进行统一介绍。

鉴于此，希赛 IT 教育研发中心组织 CSAI 顾问团有关专家，在清华大学出版社的大力支持下，编写和出版了本书，作为系统分析师考试的参考用书。期望通过本书，不但能帮助考生顺利通过考试，更重要的是帮助考生了解和掌握当前的新技术和新方法，把这些技术和方法应用到自己的工作实践中。

2. 内容

本书对当前比较前沿而又成熟的技术和方法进行了讨论，包括软件过程改进、J2EE 与.NET 平台、中间件及相关技术、应用服务器、Web Service、数据仓库、数据挖掘、商业智能、操作数据存储、异构数据库的集成、企业应用集成、XML、软件架构、设计模式、SOA、RIA、UML、UP、SOAP、PDM、PLM、AOP、P2P、工作流、软件产品线、敏捷方法、网格计算与普适计算、云计算与 SaaS、多核技术、片上系统等。这些技术和方法是任何一位合格的系统分析师必须具备的知识，也是系统分析师考试必考的知识点。阅读本书，犹如进入 IT 新技术和新方法的殿堂。

与本书的 2005 版和 2007 版相比，这次改版升级工作主要删除了一些相对陈旧的知识点，同时，对已有的内容根据读者的反馈意见进行了部分调整，增加了 2007 年以来流行和成熟的新的技术和方法。

需要说明的是，把众多的技术和方法融合在一本书中进行介绍和讨论，这是一种新的尝试和探讨，在国内尚无先例。限于篇幅，本书不可能把每种技术和方法的详细实现细节一一介绍，而是在宏观层面上对这些技术和方法进行指南性的描述。

3. 作者

本书由希赛 IT 教育研发中心组编，由张友生和王勇主编，参加写作的人员均来自 CSAI 顾问团和希赛 IT 教育研发中心。

全书共分 25 章。第 1 章由桂阳编写，第 2、4 章由戎檄编写，第 3 章由徐雷明编写，第 5、8 章由周峻松编写，第 6、7 章由张峰岭编写，第 9、10 章由王勇编写，第 11、17 章由张友生编写，第 12 章由李雄编写，第 13 章由邓子云编写，第 14 章由彭雪阳编写，第 15 章由马映冰编写，第 16 章由田俊国编写，第 18 章由徐锋编写，第 19 章由黄少华编写，第 20 章由施游编写，第 21 章由陈世帝编写，第 22、23、24 章由黄建新编写，第 25 章由尹晶海编写。

4. 致谢

在本书出版之际，要特别感谢 CSAI 顾问团的专家们，因为有了他们的无私奉献和积极参与，才使本书有可能面世。同时，本书在编写的过程中，我们参考了许多高水平的资料和书籍（详见各章的参考文献列表），在此，我们对这些参考文献的作者表示真诚的感谢。

感谢清华大学出版社柴文强老师，他在本书的策划、选题的申报、写作大纲的确定，以及编辑、出版等方面，付出了辛勤的劳动和智慧，给予了我们很多的支持和帮助。

感谢希赛教育的系统分析师学员，正是他们的想法汇成了本书的源动力，他们的意见使本书更加贴近读者。

5. 交流

由于我们水平有限，且本书涉及的知识点较多，书中难免有不妥和错误之处。我们诚恳地期望各位专家和读者不吝指教和帮助，对此，我们将深为感激。

有关本书的反馈意见，读者可在希赛教育网（http://www. educity.cn）论坛“书评在线”版块中的“希赛 IT 教育研发中心”栏目与我们交流，我们会及时地在线解答读者的疑问。

希赛 IT 教育研发中心

2009 年 3 月

目 录

第 1 章　软件过程改进

软件过程是人们建立、维护和演化软件产品整个过程中所有技术活动和管理活动的集合。目前，软件过程技术是一个非常活跃的研究领域，吸引了大批来自学术界和工业界的专家和学者。目前，每个国家几乎都有自己的软件过程改进网络和组织。软件过程技术的研究和实践主要有三个方向：

（1）软件过程分析和建模。软件过程建模方法是软件过程技术的起点，其中形式化半形式化建模方法有基于规则的，基于过程程序的等等。过程分析和过程建模对于保证过程定义的质量、建立全面和灵活的过程体系具有重要的作用。对软件过程的建模主要是使用过程建模语言（Process Modeling Languages，PML）。PML 最基本的功能是用于描述和定义过程，建立过程模型。PML 的能力和表达方式直接影响着过程模型的质量和建模效率。所以，选择合适的 PMLs，成为过程分析、过程建模和选择建模工具的关键。

（2）软件过程支持。软件过程支持主要是指研究和开发支持软件过程活动的计算机辅助软件工程（Computer-Aided Software Engineering，CASE）工具，过程支撑工具作为一种技术基础设施，能够很好地支持、管理并规范化软件过程。它的使用将使得软件过程的透明度好，为项目的软件过程提供指导，使得开发者和管理者都有据可依，便于更有效地管理软件过程。软件过程支持工具主要包括软件过程流程工具、过程文档工具、评审工具和人员管理工具。

（3）软件过程评估和改进。软件过程评估和改进是指根据某种模型对现有软件过程进行考核和评价，找出其中的不足之处，然后加以改进。改进对生产高质量软件产品和提高软件生产率的重要性已被越来越多的软件开发组织所认同。由美国卡耐基·梅隆大学软件工程研究所（CMU/SEI）提出的软件能力成熟度模型除了用于软件过程评估外，还向软件组织提供了指导其进行软件过程管理和软件过程改进的框架。软件过程改进的基本原则是采用过去项目中成功的实践经验。因此，理解、记录和重用部分软件过程是软件过程改进研究的一个重要方向。

1.1　CMM 综述

软件能力成熟度模型（Software Capacity Maturity Model，SW-CMM）是 CMU/SEI 为了满足美国联邦政府评估软件供应商能力的要求，于 1986 年开始研究的模型，并于 1991 年正式推出了 SW-CMM 1.0 版。

希赛教育专家提示：目前，有很多能力成熟度模型，例如，安全能力成熟度模型

（SE-CMM）、人力资源能力成熟度模型（P-CMM）等。在不造成混淆的情况下，本书把SW-CMM简称为CMM。也就是说，除非特别说明，否则，CMM就是指SW-CMM。

CMM自问世以来备受关注，在一些发达国家和地区得到了广泛应用，成为衡量组织软件开发和管理水平的重要参考因素，以及软件过程改进事实上的工业标准。1992年4月，CMU/SEI举行了一个CMM的研讨会，CMU/SEI在广泛听取与会专家的意见之后，于1993年推出 CMM1.1版，这也是目前世界上比较流行和通用的CMM版本。

按照CMU/SEI原来的计划，CMM的改进版本2.0应该在1997年11月完成，然后在取得版本2.0的实践反馈意见之后，在1999年完成准CMM2.0版本。但是，美国国防部办公室要求 CMU/SEI 推迟发布 CMM2.0 版本，而要先完成一个更为紧迫的项目CMMI。有关CMMI的知识，将在1.5节进行介绍。

1.1.1 CMM的基本概念

为了行文方便，在本节介绍CMM中用到的有关概念和术语。

（1）过程（process）：为实现既定目标的一系列操作步骤。

（2）软件过程（Software Process，SP）：指人们用于开发和维护软件及其相关产品的一系列活动、方法、实践和革新。其中相关产品是指项目计划、设计文档、编码、测试和用户手册。当一个组织逐步走向成熟，软件过程的定义也会日趋完善，其内部的过程实施将更具有一致性。

（3）软件过程能力（Software Process Capability，SPC）：描述了在遵循一个软件过程后能够得到的预期结果的界限范围。该指标是对能力的一种衡量，用它可以预测一个组织在承接下一个软件项目时，所能期望得到的最可能的结果。

（4）软件过程性能（Software Process Performance，SPP）：表示遵循一个软件过程后所得到的实际结果。软件过程性能与软件过程能力有区别，软件过程性能关注的是实际得到的结果，而软件过程能力关注的是期望得到的结果。由于项目要求和客观环境的差异，软件过程性能不可能充分反应软件过程整体能力，即软件过程性能受限于它的环境。

（5）软件过程成熟度（Software Process Maturity，SPM）：一个具体的软件过程被明确地定义、管理、评价、控制和产生实效的程度。所谓成熟度包含着能力的一种增长潜力，同时也表明了组织实施软件过程的实际水平。随着组织软件过程成熟度能力的不断提高，组织内部通过对过程的规范化和对成员的技术培训，软件过程也将会被它的使用者关注和不断修改完善。从而使软件的质量、生产率和生产周期得到改善。

（6）关键过程（区）域（Key Process Area，KPA）：一系列相互关联的操作活动，这些活动反映了一个软件组织改进软件过程时所必须满足的条件。也就是说，关键过程域标识了达到某个成熟程度级别时所必须满足的条件。在CMM模型中，一共有18个关键过程域，分布在第二级至第五级中。

（7）关键实践（Key Practices，KP）：关键过程域中的一些主要实践活动。每个关键

过程域最终由关键实践所组成，通过实现这些关键实践达到关键过程域的目标。一般情况下，关键实践描述了该“做什么”，但没有规定“如何”去达到这些目标。

（8）软件过程评估（Software Process Assessment，SPA）：用来判断一个组织当前所涉及的软件过程的能力状态，判断组织所面向的下一个更高层次上的与软件过程相关的课题，以及利用组织的鼎力支持来对软件过程进行有效的改进。

（9）软件能力评价（Software Capability Appraisal，SCA）：用来判断有意承担某个软件项目的组织的软件过程能力，或是判断已进行的软件过程所处的状态是否正确（或是否正常）。

（10）软件工程组（Software Engineering Group，SEG）：负责一个项目的软件开发和维护活动的团体。活动包括需求分析、设计、编码和测试等。

（11）软件相关组（Software Related Groups，SRG）：代表一种软件工程项目的团体，它支持但不直接负责软件开发或维护工作，如软件质量保证组、软件配置管理组和软件工程过程组等。在 CMM 的关键实践中，软件相关组通常应该根据关键过程域和组织的上下文来理解。

（12）软件工程过程组（Software Engineering Process Group，SEPG）：由专家组成的组，他们推进组织采用的软件过程的定义、维护和改进工作。在关键实践中，这个组织通常指“负责组织软件过程活动的组”。

（13）系统工程组（System Engineering Group，SEG）：负责下列工作的个人或团体：分析系统需求；将系统需求分配给硬件、软件和其他成分；规定硬件、软件和其他成分的界面；监控这些成分的设计和开发以保证它们符合其规格说明。

（14）系统测试组（System Test Group，STG）：一些负责策划和完成独立的软件系统测试的团体，测试的目的是为了确定软件产品是否满足对它的需求。

（15）软件质量保证组（Software Quality Assurance Group，SQAG）：一些计划和实施项目的质量保证的团体，其工作目的是保证软件过程的步骤和标准是否得到遵守。

（16）软件配置管理组（Software Configuration Management Group，SCMG）：一些负责策划、协调和实施软件项目的正式配置活动的团体。

（17）培训组（Training Group，TG）：一些负责协调和安排组织培训活动的团体。通常这个组织负责准备和讲授大多数培训课程并协调其他培训方式的使用。

1.1.2　CMM 的基本框架

CMM 模型描述和分析了软件过程能力的发展程度，确立了一个软件过程成熟程度的分级标准，如图 1-1 所示。

（1）初始级：软件过程的特点是无秩序的，有时甚至是混乱的。软件过程定义几乎处于无章法和步骤可循的状态，软件产品所取得的成功往往依赖极个别人的努力和机遇。初始级的软件过程是未加定义的随意过程，项目的执行是随意甚至是混乱的。也许，有

些组织制定了一些软件工程规范，但若这些规范未能覆盖基本的关键过程要求，且执行没有政策、资源等方面的保证时，那么它仍然被视为初始级。

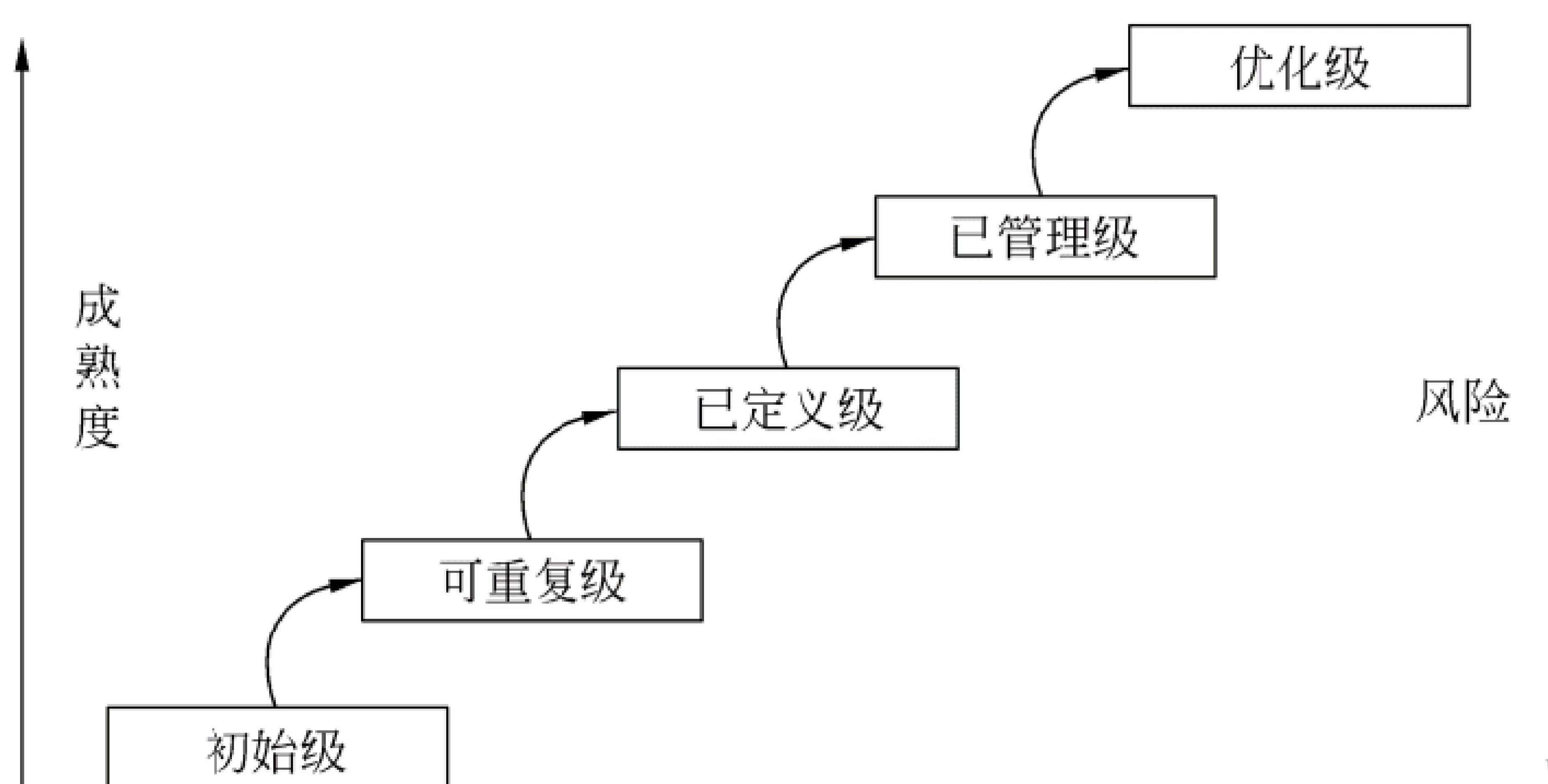


图 1-1　软件过程成熟度的级别

（2）可重复级：已经建立了基本的项目管理过程，可用于对成本、进度和功能特性进行跟踪。对类似的应用项目，有章可循并能重复以往所取得的成功。焦点集中在软件管理过程上。一个可管理的过程则是一个可重复的过程，一个可重复的过程则能逐渐演化和成熟。从管理角度可以看到一个按计划执行的且阶段可控的软件开发过程。

（3）已定义级：用于管理的和工程的软件过程均已文档化、标准化，并形成整个软件组织的标准软件过程。全部项目均采用与实际情况相吻合的、适当修改后的标准软件过程来进行操作。要求制定组织范围的工程化标准，而且无论是管理还是工程开发都需要一套文档化的标准，并将这些标准集成到组织软件开发标准过程中去。所有开发的项目需根据这个标准过程，剪裁出项目适宜的过程，并执行这些过程。过程的剪裁不是随意的，在使用前需经过组织有关人员的批准。

（4）已管理级：软件过程和产品质量有详细的度量标准。软件过程和产品质量得到了定量的认识和控制。已管理级的管理是量化的管理。所有过程需建立相应的度量方式，所有产品的质量（包括工作产品和提交给用户的产品）需有明确的度量指标。这些度量应是详尽的，且可用于理解和控制软件过程和产品，量化控制将使软件开发真正变成为一个工业生产活动。

（5）优化级：通过对来自过程、新概念和新技术等方面的各种有用信息的定量分析，能够不断地、持续地进行过程改进。如果一个组织达到了这一级，表明该组织能够根据实际的项目性质、技术等因素，不断调整软件生产过程以求达到最佳。

除第一级外，每一级都设定了一组目标，如果达到了这组目标，则表明达到了这个成熟级别，自然可以向上一级别迈进。因为从第二级开始，每一个低级别的实现均是高级别实现的基础，所以 CMM 体系不主张跨级别的演化。SEI 建议，从低一级别向高一级别演化的时间需要在 12～30 个月之间。

CMM 分级标准有两个方面的用途。一方面，软件组织利用它可以评估自己当前的过程成熟度，并以此提出严格的软件质量标准和过程改进的方法和策略，通过不断的努力去达到更高的成熟程度。另一方面，该标准也可以作为用户对软件组织的一种评价标准，使之在选择软件开发商时不再是盲目的和无把握的。

1.1.3　CMM 的主要内容

CMM 为软件组织的过程能力提供了一个阶梯式的演化框架，它采用分层的方式来解释其组成部分。在第二至第五个成熟等级中，每个等级包含一个内部结构的概念。

每一级向上一级迈进的过程中都有其特定的改进计划，具体情况如图 1-2 所示。

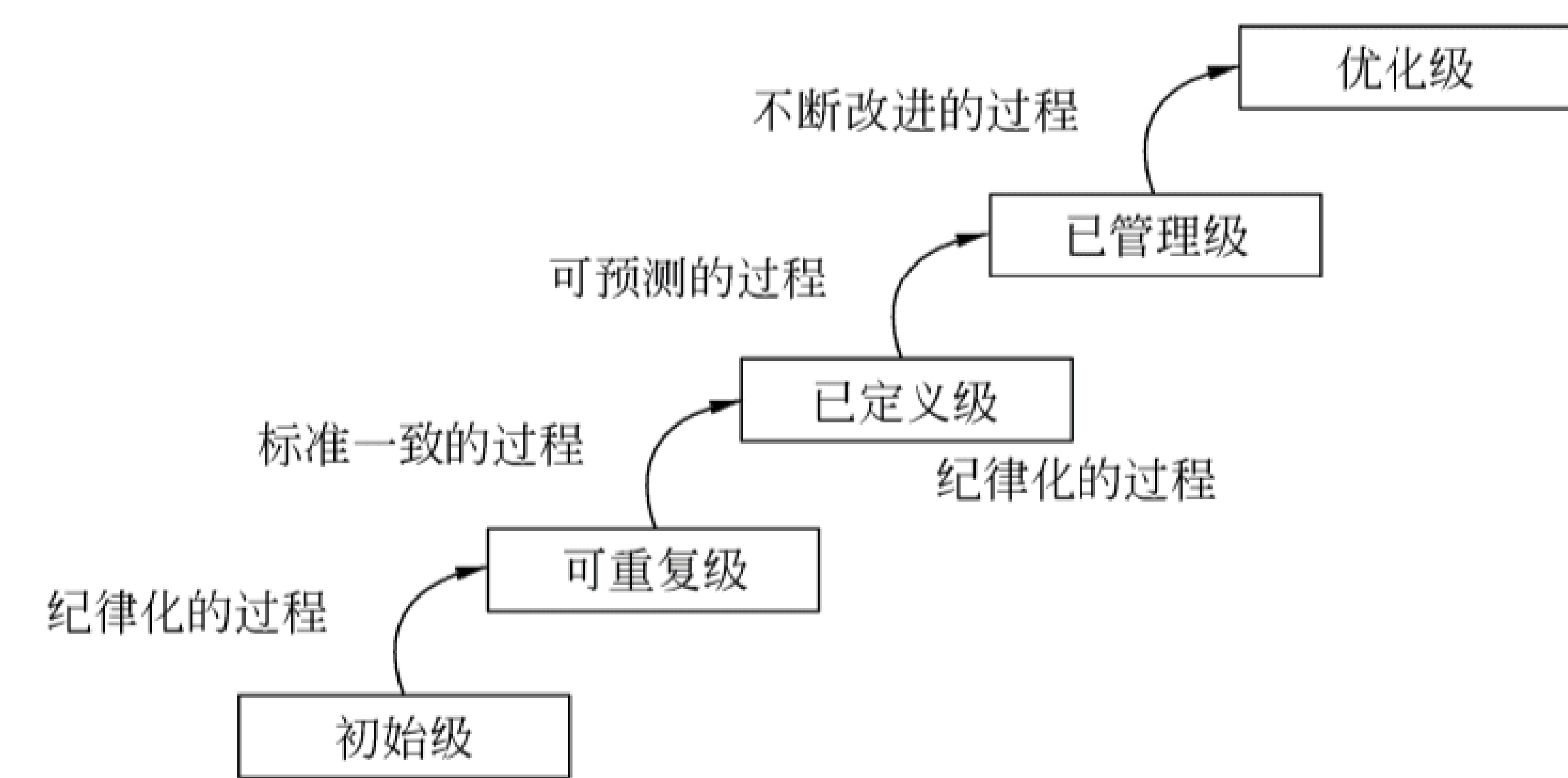


图 1-2　不断改进的过程

（1）初始级的改进方向：建立项目过程管理，实施规范化管理，保障项目的承诺；进行需求管理方面的工作，建立用户与软件项目之间的沟通，使项目真正反映用户的需求；建立各种软件项目计划，如软件开发计划、软件质量保证计划、软件配置管理计划、软件测试计划、风险管理计划及过程改进计划等；积极开展软件质量保证活动（Software Quality Assurance，SQA）。

（2）可重复级的改进方向：不再按项目制定软件过程，而是总结各种项目的成功经验，使之规则化，把具体经验归纳为组织的标准软件过程，把改进软件组织的整体软件过程能力的软件过程活动，作为软件开发组织的责任；确定全组织的标准软件过程，把软件工程及管理活动集成到一个稳固确定的软件过程中，从而可以跨项目改进软件过程，也可以作为软件过程剪裁的基础；建立 SEPG，长期承担评估与调整软件过程的任务，以适应未来软件项目的要求；积累数据，建立组织的软件过程库及软件过程相关的文档；加强培训。

（3）已定义级的改进方向：着手软件过程的定量分析，已达到定量地控制软件项目过程的效果；通过软件的质量管理达到软件质量的目标。

（4）已管理级的改进方向：防范缺陷，不仅在发现了问题能及时改进，而且应采取

特定行动防止将来出现这类缺陷；主动进行技术改革管理、标识、选择和评价新技术，使有效的新技术能在开发组织中实施；进行过程变更管理，定义过程改进的目的，经常不断地进行过程改进。

（5）优化级的改进方向：保持持续不断的软件过程改进。

1.1.4 CMM 的内部结构

CMM 的 5 个成熟度等级中，除第一级外，每一级按完全相同的内部结构构成，如图 1-3 所示。

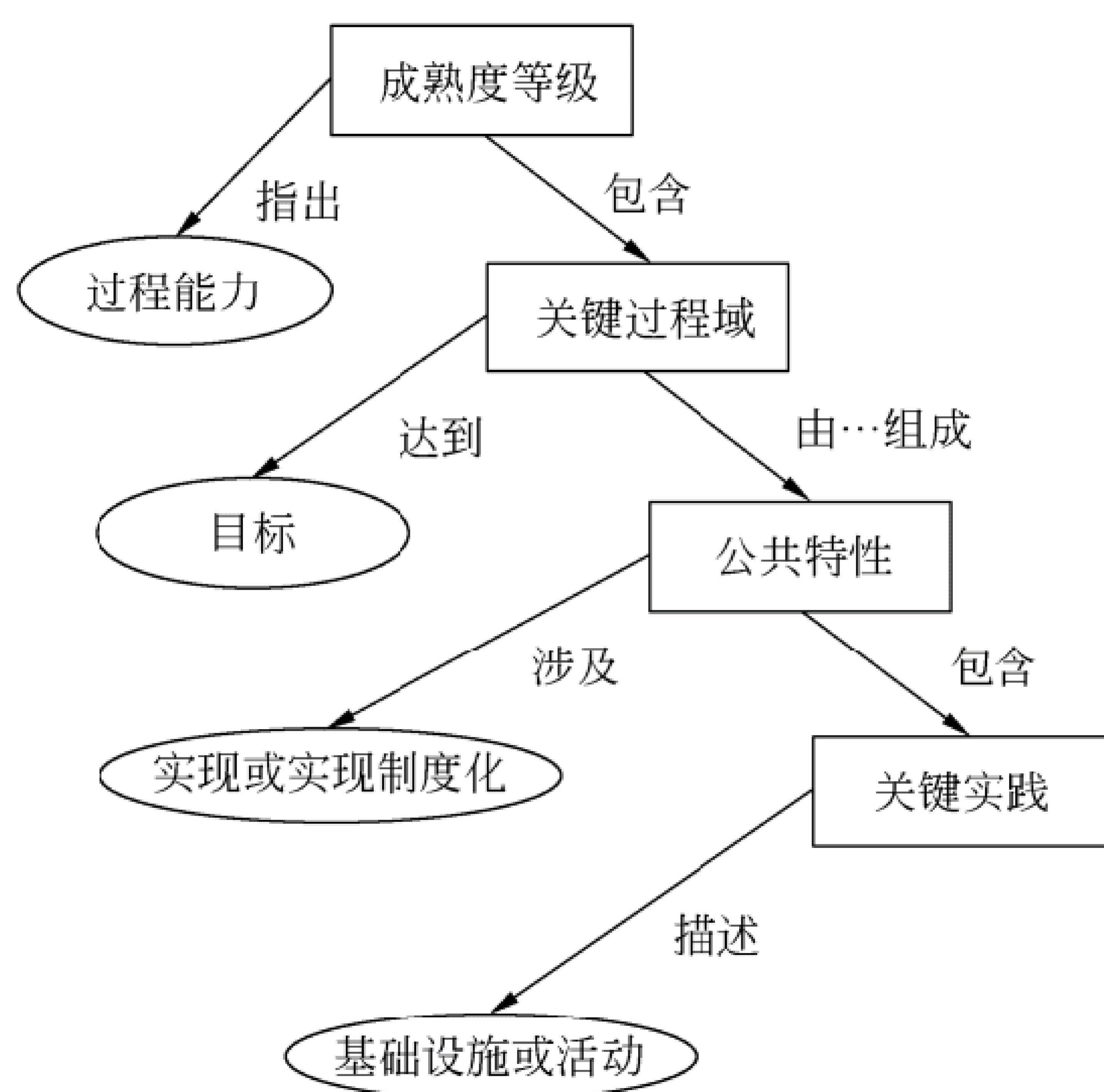


图 1-3 CMM 的内部结构图

成熟度等级为顶层，不同的成熟度等级反映了软件组织的软件过程能力和该组织可能实现预期结果的程度。

在 CMM 中，每个成熟度等级（第一级除外）规定了不同的 KPA，一个软件组织如果希望达到某一个成熟度级别，就必须完全满足 KPA 所规定的要求，即满足 KPA 的目标。每个级别对应的 KPA 如表 1-1 所示。

公共特性是把每个 KPA 的所有关键实践按照它们的属性进行分组。无论哪个 KPA，它们的关键实施都统一按 5 个公共属性进行组织，分别是执行约定、执行能力、实施活动、度量和分析、验证实施。

（1）执行约定（Commitment To Perform，CTP）：也称为实施保证，是组织为了建立和实施相应 KPA 所必须采取的行动，这些行动主要牵涉到组织范围的政策和高层管理的责任。执行约定一般与组织的方针政策和管理方式有关。

表 1-1　关键过程域的分类

过程分类 / 等级	管 理 方 面	组 织 方 面	工 程 方 面
优化级		技术改进管理 过程改进管理	缺陷预防
可管理级	定量管理过程		软件质量管理
已定义级	集成软件管理 组间协调	组织过程焦点 组织过程定义 培训程序	软件产品工程 同级评审
可重复级	需求管理 软件项目计划 软件项目跟踪与监控 软件子合同管理 软件质量保证 软件配置管理		

（2）执行能力（Ability To Perform，ATP）：也称为实施能力，描述为了使某软件过程得以始终如一地执行，必须在项目或组织中存在的先决条件，是组织实施 KPA 的前提条件。组织必须采取措施，在满足了这些条件后，才有可能执行 KPA 的实践活动。执行能力关注于项目计划的实践、资源的配置、责任的布置与授权，以及各种有关的培训等。

（3）实施活动（Activities Perform，AP）：描述执行 KPA 所需的必要行动、任务和步骤。在 5 个公共属性中，实施活动是唯一与项目执行相关的属性，其余 4 个属性则涉及组织 CMM 能力基础设施的建立。实施活动一般包括计划、执行的任务、任务执行的跟踪等。

（4）度量和分析（Measurement And Analysis，MAA）：关注 KPA 的活动需要做的度量和度量分析要求。典型的度量和度量分析的要求是确定执行活动的状态和执行活动的有效性。度量与分析一般包括一些度量的例子，通过这些例子可以知道如何确定操作活动的状态和效果。

（5）实施验证（Verifying Implementation，VI）：实施验证是验证执行活动是否与建立的过程一致，核实以确保所实施的过程是按照原定的计划以及达到其目标，着眼于保证过程的实现要通过独立的个人和高级管理人员验证。验证实施涉及到管理的评审和审计以及质量保证活动，包括过程执行的确保，产品要求的确保，高层管理人员进行的审核和项目经理进行的审核。

1.1.5　SPA 和 SCA 的比较分析

软件过程评估所针对的是软件组织自身内部软件过程的改进问题，目的在于发现缺

陷，提出改进方向。评估组以 CMM 模型为指引调查、鉴别软件过程中的问题，反过来将这些问题与 CMM 关键实践活动所提出的指导一起用于确定组织的软件过程改进策略。

软件能力评价是对接受评价者在一定条件下、规定时间内能否完成特定项目的能力考核，即承担风险的系数大小。评价包括承包者是否有能力按计划开发软件产品，是否能按预算完成等。通过利用 CMM 模型确定评价结果后，就可以利用这些结果确定选择某一承包商的风险。也可以用来判断承包者的工作进程，推动他们改进软件过程。

CMM 为评估和评价提供了一个参考框架，指出了在评估和评价中通常采用的步骤，如图 1-4 所示。

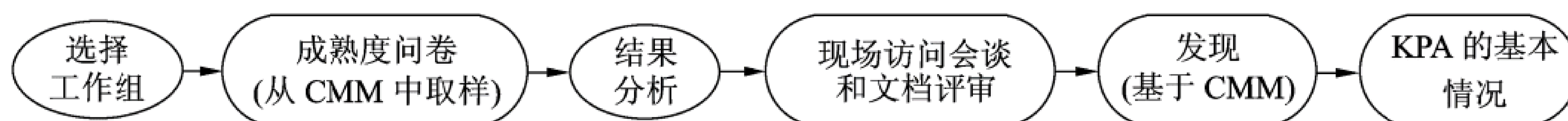


图 1-4　软件过程评估和软件能力评价的步骤

具体来说，评估过程是：选择一个工作组；完成问卷调查和取样工作；结果分析；现场访问；与 CMM 模型对照分析；依据 KPA 的基本情况列出评估提纲。

尽管 SPA 和 SCA 有很多相似之处，但由于其目的和结果的不同，它们之间的差异也是必然存在的，如：

（1）SPA 和 SCA 在出发点和目标上的不同，使得会谈目的、调查范围、收集的信息和输出的表示方式上有着本质的不同。尤其在一些细节规范方面，SPA 和 SCA 的方法有很大差异。

（2）SPA 和 SCA 的结果和结果所起的作用不同。因为两者的侧重点不一样，即使是对同一个应用项目，运用相同的方法，也不会得出相同的结果。

（3）被评估和评价单位的态度对 SPA 和 SCA 活动的影响。SPA 在某种意义上被评估单位的态度较积极，而 SCA 在某种意义上被评价单位的态度可能比较慎重。SPA 是在一个开放的、互相协作的环境中进行的，而 SCA 往往是在有较大的阻力的环境中进行的。

1.2　组织如何实施 CMM

近年来，随着国民经济持续增长，作为高新技术的软件产业虽然发展很快，但和国外同行业相比仍存在很大的差距。究其原因，投资环境、人才和技术固然是制约因素，但希赛教育专家认为，管理和政策显得更为关键。随着电子信息产业的发展，人们已经逐步认识到，软件是促进我国电子信息产业发展的关键技术。而要发展我国的软件产业，在战略上，必须将软件产业作为我国高新技术产业的龙头和国民经济发展的新增长点，

在策略上，必须走软件过程管理专业化的道路。

1．提高思想认识

实施 CMM 对软件组织的发展起着至关重要的作用，这种作用并不是取决于 CMM 评估的结果。CMM 过程本身就是对软件组织发展历程的一个完整而准确的描述，组织通过实施 CMM，可以更好地规范软件生产和管理流程，使组织的管理更加规范化。而且，只有在国际市场取得成功的产品和组织才具有长久的竞争力和生命力，由于 CMM 已获得国际组织和用户的广泛认可，因此很有必要在软件组织中实施 CMM。

2．进行 CMM 培训和咨询工作

任何一个组织要想实施先进的管理措施，首先应该做的就是理论基础的建设，作为一个过程式管理方法的 CMM，同样也不例外。

根据 CMM 模型的要求，一个项目的开发一定要有章可循，而且要做到有章必循，这两点都离不开培训。培训工作需要投入很大的人力、物力和财力，只有组织的管理人员和软件开发人员对 CMM 真正了解和认识了，自觉地按 CMM 的方法去进行工作，才能真正实施 CMM，而不是一时应付，做表面文章。

培训的内容需要精心地准备，主要有两个方面，第一，对所有员工包括经理在内的最基本的软件工程和 CMM 培训知识；第二，对各个工作组的有关人员提供专业领域知识等方面的培训；此外，在每次开发过程中，还要对普通人员进行软件过程方面的培训。

培训的方式有很多，可以向有关专业培训咨询机构进行咨询；可以利用互联网资源进行咨询和培训；可以聘请有关 CMM 专家到组织实地指导 CMM 的实施。

希赛教育专家提示：组织可以在最开始阶段聘请一位经验丰富的 CMM 专家，但以后一定要培养自己的专家，这样不仅能节约开支，还能使组织自己具有一个对 CMM 深刻理解的、有实践经验的专家，为组织今后的继续升级打下一个良好的基础。

3．确定合理的目标

CMM 模型划分为 5 个级别，共计 18 个关键过程域，52 个目标，316 个关键实践。每一个 CMM 等级的评估周期（从准备到完成）约需 12～30 个月。无论一个组织的软件过程处于什么样的水平，都可以在 CMM 框架的 5 个级别中找到自己的位置。CMM 框架的不同级别是针对处于不同管理水平的组织制定的，组织实施 CMM，首先必须了解自己的管理现状，对照 CMM 的级别，找到自己在 CMM 中所处的位置，然后有针对性采取与自己所处级别相适应的措施，使组织尽早纳入 CMM 的演化阶段，使软件过程管理早日得到改善，最终达到提高软件质量，获取经济效益的目的。

因此，要实施 CMM，首先应该对本组织的现状有一个准确的评估。组织目前处于什么水平，组织发展的问题是什么，借助 CMM 要达到的目的是什么。然后再结合组织的实际情况选择 CMM 的切入点，确定总体目标。这个目标包括在多长时间之内，需要投入多少人力、物力和财力，要达到哪一级。

由于软件过程的建立和改进是一个渐进的、分轻重缓急的、逐步完善的过程。所以，

在总体目标已经确定的前提下，还要制订近期目标和长期目标。

4．成立工作组

组织针对 CMM 的实施，应成立专门的 CMM 实施领导小组或专门的机构。CMM 的实施需要有强有力的组织保证，领导层必须真正学习理解软件过程管理和改进的重要性，亲自领导和参与，要保证过程管理的人员配备，抽调组织中有管理能力、组织能力和软件开发能力的骨干人员，确实把此项工作当作组织生存和发展的大事来抓。

在 CMM 的实施过程中，工作组的成立是 CMM 的一个关键步骤。有几个重要的组织是必不可少的，这些组织包括软件工程过程组、软件工程组、系统工程组、系统测试组、需求管理组、软件项目计划组、软件项目跟踪与监督、软件配置管理组、软件质量保证组、培训组。

在 CMM 的实施中组织机构的设置必须完善，但不等于说每一个机构必须是独立的。有些组织很小时，机构可以适当合并，成员可以身兼数职。但对那些关键实践要求独立性时，组织必须十分小心。例如，软件质量保证组的独立性就是必须考虑的，否则在技术上或机构上出现的偏差，会无目的地影响到软件过程、项目质量和风险决策的正确性。

在这里还要提到一点，那就是物理组和逻辑组。在 CMM 中有两种组织，一种叫物理组织，它是客观存在的，例如项目组、技术部等，有众多专职人员；另一种叫逻辑组织，就是说它的人员可以是兼职的，在用不到的时候，成员有自己的工作，而且很多逻辑组只需一两个人就可以了。

5．制定和完善软件过程

CMM 模型强调软件过程的改进，如果组织还没有一个文档形式的软件过程，则首要任务是对当前的工作流程进行分析、整理及文档化，从而制定出一个具有本组织风格的软件过程，并用该文档化的过程指导软件项目的开发。

如果已经具备了软件过程，则要对这个过程做内部评估，对照 CMM 的要求，找出问题，然后对这个过程进行补充修改。在具体实施的过程中，可以选择有一定代表性和完善性的项目组或项目进行试点，跟踪、监督改进后的软件过程的实施情况，执行改进活动的状态。

同时，过程小组的成员还应该维护过程中的数据库，定期统计各个过程中的产品和规模、开发周期、修改次数及评估周期。这些数据库可用来分析项目的效率以及存在的问题，以便今后进一步的改进，同时还可以为项目开发过程提供咨询。

总结这些项目组或项目以前成功的经验，从中规划出一个具有实际意义的软件过程，按照 CMM 规范评估这个过程，找出其中的优缺点。对不满足 CMM 要求的地方加以完善，使其成为一个完美的实施 CMM 的软件过程方案；然后将这个软件过程应用到当前正在承接的或即将承接的项目上，在实际使用过程中进一步发现其中的不足和错误之处，加以改进，最后将试点的结果推广到整个组织。

6．内部评审

CMM 每一级别的评估都由 CMU/SEI 授权的主任评估师领导一个评审小组进行，相

对来说，评审费用比较高。因此，希赛教育专家建议，组织在进行正式评估之前，先进行内部评审或评估。组织自己内部成员，严格、认真地按照 CMM 规范评估过程，对自己的软件过程进行评审，找出其中的不足点并进行改进。这样，就能确保正式评估时能一次性通过，节约成本。

7．正式评估

CMM 正式评估的过程包括员工培训（组织的高层领导也要参加）、问卷调查和统计、文档审查、数据分析、与组织的高层领导讨论和撰写评估报告等，评估结束时由主任评估师签字生效。

基于 CMM 的评估方法主要有两种，一种是 CBA-SCE（CMM-Based Appraisal for Software Capability Estimation），它是基于 CMM 对组织的软件能力进行评价，是由组织外部的评估小组对该组织的软件能力进行的评价。另一种是 CBA-IPI（CMM-Based Appraisal for Internal Process Improvement），它是基于 CMM 对内部的过程改进进行的评估，是由组织内部的小组对软件组织本身进行评估以改进质量，结果归组织所有，目的是引导组织不断改进质量。

这两种评估均由 CMU/SEI 授权的主任评估师领导，参考 CMM 框架来进行，都要审查正在使用和将来使用的文档，并对不同的员工进行采访。SCE 与 IPI 的评估结果应该一致，评估结果的所有资料都将呈报给 CMU/SEI。

8．根据评估结果改进软件过程

根据 CMM 模型，成熟度的评估只是软件过程改进中的一个环节，如果这个环节与软件过程改进的其他环节不能很好地结合，那么，CMM 评估对于软件过程改进所应具有的作用就得不到发挥。

一般来说，应该在评估之后很快地作出软件过程改进的计划，因为这时大家对评估结果和存在的问题仍有一个深刻的认识。计划在软件过程改进中是一个非常必要的阶段，只有有效的计划，才能确保软件过程得到有效的改进。

CBA 评估方法对衡量软件组织的能力成熟度是一个非常有效的手段，评估结果本身就是一个非常坚实的基础，是制定软件过程改进计划的依据。CBA 评估客观地指出了组织软件过程存在的问题，帮助组织发现软件过程的不足之处，充分指出了软件过程改进的前景。

1.3　CMM 存在的问题

从 CMM 评估的实践来看，CMM 1.1 版主要存在以下问题。

（1）CMM 虽然指明了成熟的软件过程的各种关键实践，并提供了一些有效的实践例子，但 CMM1.1 并不包括对成功的项目来说是必不可少的一些重要问题，包括人才、个人技能和具体的技术等。

（2）CMM 1.1 所描述的标准和实践很适宜于与政府签约的大型软件开发组织和大项目，但是对于中小型组织或中小项目来说，必需加以适当剪裁。但是，CMM 1.1 却并没有给出操作性好的剪裁指南。

（3）CMM 1.1 发布时，关于软件生存周期过程的国际标准（ISO/IEC 12207-1995）尚未发布，国际标准化组织（International Standard Organization，ISO）也刚确定要制定软件过程评估的国际标准。因此，CMM 1.1 所涉及的过程与 ISO/IEC 12207 对于相应的过程的阐述不完全一致；在关于软件过程评估工作的技术文件中，CMM 1.1 与 ISO 的表述也有差异。

（4）CMM 1.1 关于等级 4 和等级 5 的阐述，在实际经验方面的依据还不足，对这类组织的特征了解较少，因此，有关的 KPA 及所包含的关键实践的定义不像等级 2 和等级 3 的 KPA 的定义那样完善和明晰。

（5）CMM 1.1 没有指明任一软件开发组织都必须首先建立基本的软件工程过程和管理过程，否则就较难从初始级顺利地提高到可重复级。在我国的实践也表明，从初始级到可重复级的台阶太高，不利于软件开发组织从初始级演化到可重复级。

1.4 ISO 9001 与 CMM 的比较

表 1-2 是 ISO 9001 条款到 CMM 模型 KPA 和关键实践映射的概述。“强相关性”列表示相关性较直接的 KPA 和共同特征；“判断相关性”列表示在确定合理相关性时需要一定程度上的主观理解的 KPA 和共同特征。

表 1-2 ISO 9001 到 CMM 的映射

ISO 9001 条款	强 相 关 性	推断相关性
管理职责	履行的承诺 软件项目规划 软件项目追踪和监督 软件质量标准	履行的能力 实现矫正 软件质量管理
质量体系	实现矫正 软件项目规划 软件质量标准 软件产品工程	组织工程定义
合同评估	需求管理 软件项目规划	软件子合同管理
设计控制	软件项目规划 软件项目追踪和监督 软件配置管理 软件产品工程	软件质量管理

续表

ISO 9001 条款	强 相 关 性	推断相关性
文档和数据控制	软件配置管理 软件产品工程	
采购	软件子合同管理	
客户－供货产品的控制		软件子合同管理
产品确认和追踪	软件配置管理 软件产品工程	
工程控制	软件项目规划 软件质量帮助 软件产品工程	定量工程管理 技术改变管理
检查和测试	软件产品工程 伙伴审查	
检查控制、度量和测试设备	软件产品工程	
检查和测试状态	软件配置管理 软件产品工程	
不合格产品的控制	软件配置管理 软件产品工程	
矫正和预防措施	软件质量保证 软件配置管理	缺陷预防
处理、储藏、包装、保存和分发		软件配置管理 软件产品工程
质量数据控制	软件配置管理 软件产品工程、伙伴审查	
内部质量审计	实现检查、软件质量保证	
培训	履行的能力、培训计划	
服务		
统计技术	度量和分析	机构过程定义 定量过程管理 软件质量管理

虽然 ISO 9001 中的一些问题没有被 CMM 模型覆盖，二者之间的详细程度也有很大的差异，但二者之间的相关性还是很明显的。CMM 与 ISO 9001 之间最大的不同体现在两方面：

第一，CMM 模型明确强调持续的过程改进，而 ISO 9001 只要求质量体系的最小保证（希赛教育专家提示：ISO 9001 2000 版也引进了“持续的过程改进”）。

第二，CMM 模型只关注软件，而 ISO 9001 适用于更大的范围。ISO 9001 的基本假设是：组织应该通过质量控制活动，归档每个重要过程并检查每个重要过程。CMM 模

型也强调文档化的过程和文档化的设计。“按文档化的程序”和遵循“书面形式的组织政策”是 CMM 模型 KPA 的特征。

通过以上分析，可以得到以下结论：

（1）ISO 9001 和 CMM 既有区别又相互联系。尽管 ISO 9001 标准的一些要求在 CMM 中不存在，而 CMM 的一些要求在 ISO 9001 标准中也不存在，但不可否认的是，两者之间的关系非常密切。当然，两者之间的差别也很明显。ISO 9001 的一些要素可以在 CMM 中找到完全对应的部分，另外一些要素则是比较分散的对应。

（2）取得 ISO 9001 认证并不意味着完全满足 CMM 某个等级的要求。表面上看，获得 ISO 9001 认证的组织应该具有 CMM 第 3 级至第 4 级的水平，但事实上，很多 CMM 第 1 级的组织也获得了 ISO 9001 证书，原因是 ISO 9001 强调以顾客的要求为出发点，不同顾客要求的质量水平也不同，而且各个审核员的水平和解释也有差异。ISO 9001 标准只是质量管理体系的最低可接受准则，不能说已满足 CMM 的大部分要求，但有一点可以肯定：ISO 9001 认证合格的组织至少能满足 CMM 第 2 级的大部分要求以及第 3 级的一部分要求。

（3）通过 CMM 第 2 级（或第 3 级）评估并不代表满足 ISO 9001 的要求。CMM 第 2 级的所有关键过程都涉及 ISO 9001 的要求，但都低于 ISO 9001 的要求。另外，一些 CMM 第 1 级的组织在满足了第 2 级和第 3 级的一些关键过程的要求后，也可以获得 ISO 9001 认证。一些 CMM 第 2 级或第 3 级的组织可能被认为符合 ISO 9001 的要求，但是，甚至一些通过了 CMM 第 3 级评估的组织也需另外满足 ISO 9001 的要素，才能符合 ISO 9001 的要求。

总的来说，CMM 是专门针对软件开发组织设计的，因此在针对性上比 ISO 9001 要好，但需要注意的是，CMM 强调的是软件开发过程的管理，对于国内软件组织涉及较多的系统集成并没有考虑，如果单纯按照 CMM 的要求建立质量体系，则应该注意补充系统集成方面的内容。正因为这个原因，CMU/SEI 在 CMM 模型的基础上，综合考虑了多个模型，开发了 CMMI 模型。

1.5 CMMI 综述

能力成熟度模型集成（Capability Maturity Model Integration，CMMI）是 CMM 模型的最新版本，2001 年 12 月，CMU/SEI 正式发布了 CMMI 1.1 版本。与原有的能力成熟度相比，CMMI 涉及面更广，专业领域覆盖软件工程、系统工程、集成产品开发和系统采购。据美国国防部资料显示，运用 CMMI 模型管理的项目，不仅降低了项目的成本，而且提高了项目的质量与按期完成率。

CMMI 可以看作是把各种 CMM 集成到一个系列的模型中，CMMI 的基础源模型包括 SW-CMM 2.0 版（草稿 C），EIA-731 系统工程，以及集成化产品和过程开发 IPD-CMM

0.98a 版。CMMI 也描述了 5 个不同的成熟度级别。

1. CMMI 模型的表示

每一种 CMMI 模型都有两种表示法，分别是阶段式和连续式。这是因为在 CMMI 的三个源模型中，SW-CMMM 是阶段式模型，EIA-731 是连续式模型，而 IPD-CMM 是一个混合模型，结合了阶段式和连续式两者的特点。两种表示法在以前的使用中各有优势，都有很多支持者，因此，CMMI 产品开发组在集成这三种模型时，为了避免由于淘汰其中任何一种表示法而失去对 CMMI 支持的风险，并没有选择单一的结构表示法，而是为每一个 CMMI 都推出了两种不同表示法的版本。

不同表示法的模型具有不同的结构。连续式表示法强调的是单个过程域的能力，从过程域的角度考察基线和度量结果的改善，其关键术语是“能力”；而阶段式表示法强调的是组织的成熟度，从过程域集合的角度考察整个组织的过程成熟度阶段，其关键术语是“成熟度”。

尽管两种表示法的模型在结构上有所不同，但 CMMI 产品开发组仍然尽最大努力确保了两者在逻辑上的一致性，两者的需要构件和期望部件基本上都是一样的。过程域、目标在两种表示法中都一样，特定实践和共性实践在两种表示法中也不存在根本性的区别。因此，模型的两种表示法并不存在本质上的不同。组织在进行集成化过程改进时，可以从实用角度出发选择某一种偏爱的表示法，而不必从哲学角度考虑两种表法之间的差异。

阶段式模型也把组织分为 5 个不同的级别：

（1）初始级。以不可预测结果为特征的过程成熟度，过程处于无序状态，成功主要取决于团队的技能。

（2）已管理级。以可重复项目执行为特征的过程成熟度。组织使用基本纪律进行需求管理、项目计划、项目监督和控制、供应商协议管理、产品和过程质量保证、配置管理，以及度量和分析。对于已管理级而言，主要的过程焦点在于项目级的活动和实践。

（3）严格定义级。以组织内改进项目执行为特征的过程成熟度。强调严格定义级的 KPA 的前后一致的、项目级的纪律，以建立组织级的活动和实践。

（4）定量管理级。以改进组织性能为特征的过程成熟度。定量管理级项目的历史结果可用来交替使用，在业务表现的竞争尺度（成本、质量、时间）方面的结果是可预测的。

（5）优化级。以可快速进行重新配置的组织性能，和定量的、持续的过程改进为特征的过程成熟度。

2. CMMI 的目标和优点

CMMI 的具体目标如下：

（1）改进组织的过程，提高对产品开发和维护的管理能力。

（2）给出能支持将来集成其他科目 CMM 的公共框架。

（3）确保所开发的全部有关产品符合软件过程改进的国际标准 ISO/IEC15504 对软件过程评估的要求。

使用在 CMMI 框架内开发的模型具有下列优点：

（1）过程改进能扩展到整个组织级。

（2）以前各模型之间的不一致和矛盾将得到解决。

（3）既有分级的模型表示，也有连续的模型表示，任组织选用。

（4）原先单方面（例如软件）过程改进的工作可与其他方面（例如安全、系统集成等）的过程改进工作结合起来。

（5）基于 CMMI 的评估将与组织 CMM 评估得分相协调，从而保护当前的投资。并与 ISO/IEC15504 评估结果相一致。

（6）节省费用，特别是当要进行多个方面的改进时，以及进行相关的培训和评估时。

（7）鼓励组织内各方面之间进行沟通和交流。

3. CMMI 评估

CMMI 产品集中的 CMMI Appraisal Requirement version1.1 给出了基于 CMMI 的评估中 40 多条需求，提供了一个综合需求集和评估方法的设计限制。根据这些需求集和设计限制，可以开发出相应的基于 CMMI 的评估方法。

CMMI 产品集中还给出了过程改进的标准方法——CMMI 评估方法，这一方法满足全部的 CMMI Appraisal Requirement version 1.1 需求。组织在进行过程改进时可以参考该方法进行具体实施。

但是，对于一个具体的组织来说，仅仅根据 CMMI 产品中对评估的相应描述来照本宣科是远远不够的。特别是对于一些在过程管理方面还很不成熟的组织，在开展 CMMI 评估的时候更要发挥自身的创造性，灵活运用 CMMI 产品集中所提供的评估方法。根据 CMMI Appraisal Requirement version 1.1 中的描述，评估可分为三类。

（1）A 类评估：全面综合的评估方法，要求在评估中全面覆盖评估中所使用的模型，并且在评估结果中提供对组织的成熟度等级的评定结果。

（2）B 类评估：较少综合，花费也较少。在开始时作部分自我评估，并集中于需要关注的过程域。不评定组织的成熟度等级。

（3）C 类评估：也称为快估。主要是检查特定的风险域，找出过程中存在的问题。这类评估花费很少，需要的培训工作也不多。

1.6 CMM 与 CMMI 的比较

CMMI 阶段式的基本结构从 CMM 演变而来，但是 CMMI 的结构更加形式化和精致，也更加复杂，尤其为了保证连续式和阶段式的统一性，增加了结构的理解难度。

CMMI 强调了对需求的管理，有两个 KPA 说明对需求的控制，分别是需求管理和需

求开发。而在 CMM 中只有一个 KPA（需求管理）以及软件产品工程中的一个实践来说明对需求的管理和控制。

CMMI 加强了对工程过程的重视，提供了更加细致的要求和指导，而 CMM 中却只有一个 KPA（软件产品工程）来进行要求和指导。

CMMI 强调了度量，并且从项目的早期就已经进行了度量，在阶段式中 CMMI 二级有一个过程域度量和分析，而在 CMM 中没有专门的要求和指导。

CMMI 比 CMM 更加强调了对风险的管理，在 CMM 中风险只是项目策划中的一个活动，而在 CMMI 中风险管理作为一个单独的过程域。

CMM 作为较早推出的软件过程改进标准，目前已在世界范围被广泛地推广和应用。不可否认，CMM 在建立有效的开发系统、控制软件产品开发过程、降低软件组织内外部故障成本、提高组织的经济效益和社会效益等方面，取得了巨大成功。但 CMM 也存在着一些不足，例如，CMM 中的一个 KPA（组间协调）在 CMMI 中地位下降，只是作为集成化项目管理中的一个目标。

CMM 中的同行评审，在 CMMI 中得到了更高的抽象，对应 CMMI 的验证，说明了对产品进行相应的质量保证活动。

在 CMMI 的公共特性中，没有了测量，这些度量内容被组织起来形成了一个支持过程（度量和分析）。具体理由如下：

度量和分析本身应用的复杂性和它执行的高成本在原来的 CMM 中每个 KPA 均有单独的测量要求，容易造成过度测量，也没有形成对组织级的、统一的度量体系的指导和要求，造成实施中的困难。例如，在 CMM 中如果一个组织达到了 3 级，由于各个 KPA 均要求了测量，实际上已经建立了全组织过程的测量，这和 CMM 的等级划分思想是有冲突的。

CMMI 改进了这个方面，要求组织从组织级的统一要求出发建立度量体系。这样的想法也符合过程改进理论的思想。这样，组织在实施过程中可以选择必要的过程进行测量，而不是全部过程的测量，从这个意义上，CMMI 比 CMM 降低了对度量的要求和实施难度，但是更加具有全局性和可实施性。

最后，CMM 是作为评估标准出现的，所规定的内容都是必要的，这样才能保证评估的标准；而 CMMI 是作为改进模型出现的，罗列了较多的最佳实践，以利于过程的改进。

1.7　个体软件过程

随着软件工程知识的普及，软件工程师都知道，要开发高质量的软件，必须改进软件生产的过程。目前，业界公认由 CMU/SEI 开发的 CMM/CMMI 是当前最好的软件过程，并且已经成为事实上的软件过程工业标准。但是，CMM/CMMI 虽然提供了一个有

力的软件过程改进框架，却只告诉“应该做什么”，而没有告诉“应该怎么做”，并未提供有关实现 KPA 所需要的具体知识和技能。

为了弥补这个缺陷，CMU/SEI 高级成员和研究科学家 Watts S.Humphrey 博士又主持开发了个体软件过程（Personal Software Process，PSP）。PSP 是一种可用于控制、管理和改进个人工作方式的自我持续改进过程，是一个包括软件开发表格、指南和规程的结构化框架。PSP 与具体的技术（程序设计语言、工具或设计方法）相对独立，其原则能够应用到几乎任何的软件工程任务之中。PSP 能够说明个体软件过程的原则，帮助软件工程师作出准确的计划，确定软件工程师为改善产品质量要采取的步骤，建立度量个体软件过程改善的基准，确定过程的改变对软件工程师能力的影响。

在 CMM1.1 版本的 18 个 KPA 中有 12 个与 PSP 有关，据统计，软件项目开发成本的 70%取决于软件开发人员个人的技能、经验和工作习惯。因此，一个组织的软件开发人员如果能接受 PSP 培训，对该组织软件能力成熟度的升级是一个有力的保证。CMM/CMMI 侧重于软件组织中有关软件过程的宏观管理，面向整个软件开发组织，PSP 则侧重于组织中有关软件过程的微观优化，面向软件开发人员。二者互相支持，互相补充，缺一不可。

就像 CMM/CMMI 为软件组织的能力提供一个阶梯式的演化框架一样，PSP 为个体的能力也提供了一个阶梯式的演化框架，以循序渐进的方法介绍过程的概念，每一级别都包含了更低一级别中的所有元素，并增加了新的元素。这个演化框架是学习 PSP 过程基本概念的好方法，它赋予软件人员度量和分析工具，使其清楚地认识到自己的表现和潜力，从而可以提高自己的技能和水平。PSP 演化框架共有四级，各级及其增强版的主要元素如图 1-5 所示。

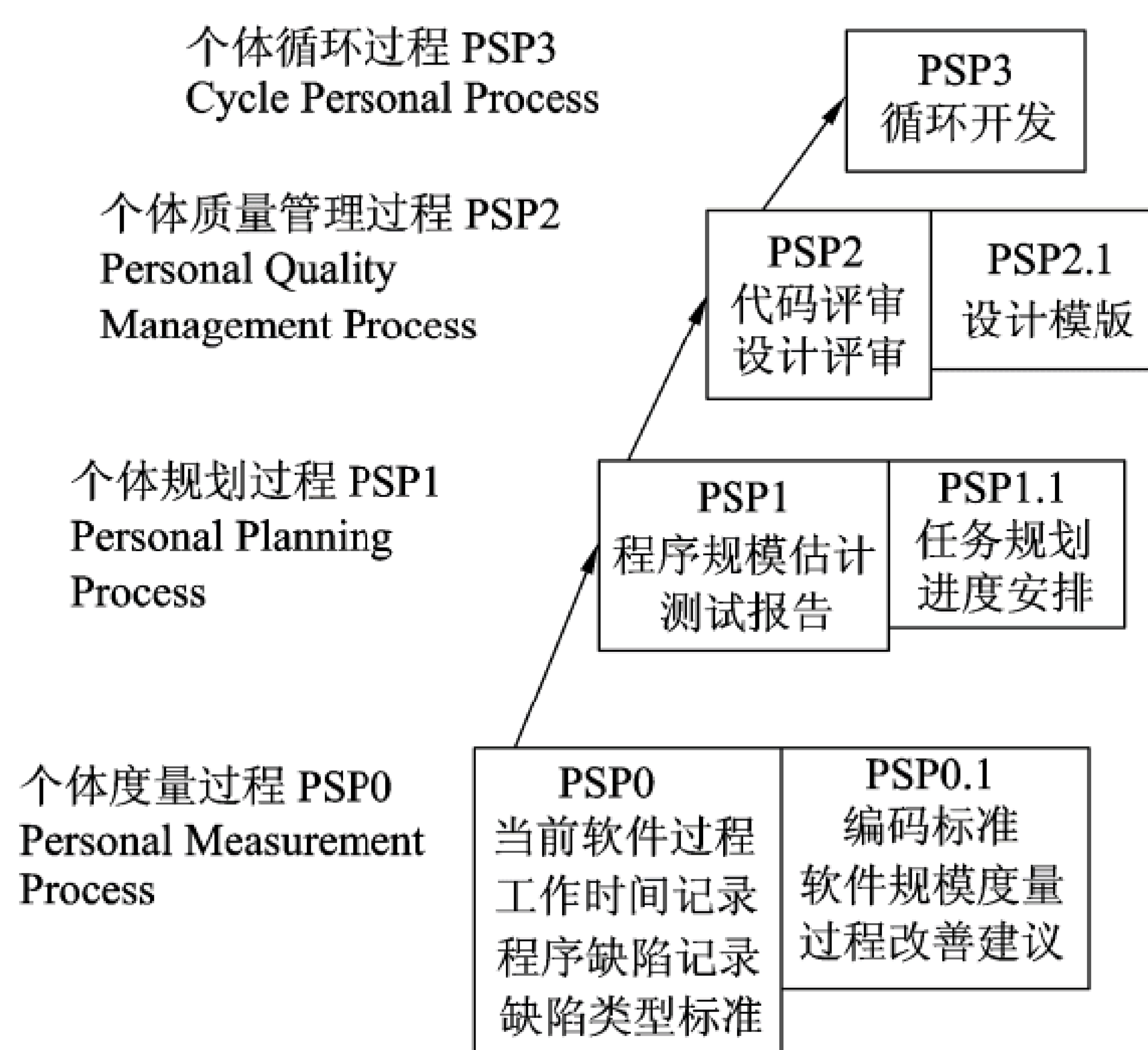


图 1-5　PSP 的演化框架图

1．个体度量过程

个体度量过程包括 PSP0 和 PSP0.1。

PSP0 的目的是建立个体过程基线，通过这一步，学会使用 PSP 的各种表格采集过程的有关数据，此时执行的是该软件开发单位的当前过程，通常包括计划、开发（包括设计、编码、编译和测试）以及后置处理三个阶段，并要作一些必要的试题，如测定软件开发时间，按照选定的缺陷类型标准、度量引入的缺陷个数和排除的缺陷个数等，用作测量在 PSP 的过程中进步的基准。

PSP0.1 增加了编码标准、程序规模度量和过程改善建议等三个关键过程域，其中过程改善建议表格用于随时记录过程中存在的问题、解决问题的措施以及改进过程的方法，以提高软件开发人员的质量意识和过程意识。

希赛教育专家提示：在 PSP0 阶段，软件工程师必须理解和学会使用进行规划和度量的技术。设计一个好的表格并不容易，需要在实践中积累经验，以准确地满足期望的需求，其中最重要的是要保持数据的一致性、有用性和简洁性。

2．个体规划过程

个体规划过程包括 PSP1 和 PSP1.1。

PSP1 的重点是个体计划，引入了基于估计的计划方法 PROBE（PROxy Based Estimating），用自己的历史数据来预测新程序的大小和需要的开发时间，并使用线性回归方法计算估计参数，确定置信区间以评价预测的可信程度。PSP1.1 增加了对任务和进度的规划。

在 PSP1 阶段，软件工程师应该学会编制项目开发计划，这不仅对承担大型软件的开发十分重要，即使是开发小型软件也必不可少。因为，只有对自己的能力有客观的评价，才能作出更加准确的计划，才能实事求是地接受和完成客户（顾客）委托的任务。

3．个体质量管理过程

个体质量管理过程包括 PSP2 和 PSP2.1。

PSP2 的重点是个体质量管理，根据程序的缺陷建立检测表，按照检测表进行设计复查和代码复查（代码走查），以便及早发现缺陷，使修复缺陷的代价最小。随着个人经验和技术的积累，还应学会怎样改进检测表以适应自己的要求。PSP2.1 则论述设计过程和设计模板、介绍设计方法，并提供了设计模板。但是，PSP 并不强调选用什么设计方法，而强调设计完备性准则和设计验证技术。

实施 PSP 的一个重要目标就是学会在开发软件的早期实际地、客观地处理由于人们的疏忽所造成的程序缺陷问题。人们都期盼获得高质量的软件，但是只有高素质的软件开发人员并遵循合适的软件过程，才能开发出高质量的软件，因此，PSP2 引入并着重强调设计复查和代码复查技术，一个合格的软件开发人员必须掌握这两项基本技术。

4．个体循环过程

个体循环过程只包括 PSP3。

PSP3 的目标是把个体开发小程序所能达到的生产效率和生产质量，延伸到大型程序。其方法是采用螺旋式上升过程，即迭代增量式开发方法，首先把大型程序分解成小的模块，然后对每个模块按照 PSP2.1 所描述的过程进行开发，最后把这些模块逐步集成为完整的软件产品。

应用 PSP3 开发大型软件系统，必须采用增量式开发方法，并要求每一个增量都具有很高的质量。在这样的前提下，在新一轮开发循环中，可以采用回归测试的方法，集中力量考察新增加的这个（这些）增量是否符合要求。因此，要求在 PSP2 中进行严格的设计复查和代码复查，并在 PSP2.1 中努力遵循设计结束准则。

从对个体软件过程框架的概要描述中，可以清楚地看到，如何做好项目规划和如何保证产品质量，是任何软件开发过程中最基本的问题。

PSP 可以帮助软件工程师在个人的基础上运用过程的原则，借助于 PSP 提供的一些度量和分析工具，了解自己的技能水平，控制和管理自己的工作方式，使自己日常工作的评估、计划和预测更加准确、更加有效，进而改进个人的工作表现，提高个人的工作质量和产量，积极而有效地参与组织范围的软件过程改进。

1.8 团队软件过程

CMU/SEI 在 1991 年提出了 CMM，1994 年提出 PSP，1999 年提出了团队软件过程（Team Software Process，TSP），使软件过程框架形成一个包含 CMM、PSP 和 TSP 三者的严密的整体。

1.8.1 TSP 概述

TSP 对团队软件过程的定义、度量和改进提出了一整套原则、策略和方法，把 CMM 要求实施的管理与 PSP 要求开发人员具有的技巧结合起来，以按时交付高质量的软件，并把成本控制在预算的范围之内。在 TSP 中，讲述了如何创建高效且具有自我管理能力的开发团队，软件工程师如何才能成为合格的项目组成员，管理人员如何对团队提供指导和支持，如何保持良好的工程环境使项目组能充分发挥自己的水平等软件工程管理问题。

1. TSP 的基本原理

TSP 基于以下 4 条基本原理：

（1）应该遵循一个确定的、可重复的过程并迅速获得反馈，这样才能使学习和改进最有成效。

（2）一个团队是否有效，是由明确的目标、有效的工作环境、有能力的教练和积极的领导这 4 个方面的因素综合作用所确定的，因此，团队应该在这 4 个方面同时努力，而不能偏废其中任何一个方面。

（3）应注意及时总结经验教训，当软件工程师在项目中面临各种各样的实际问题并寻求有效的解决问题方案时，就会更深刻地体会到 TSP 的威力。

（4）应注意借鉴前人和他人的经验，在可以利用的工程、科学和教学法经验的基础上，规定过程改进的流程。

2．实施 TSP 的先决条件

组织要实施 TSP，需要具备 5 个先决条件：

（1）需要有高层主管和各级管理人员的支持，以取得必要的资源，这是实施 TSP 必须具备的物质基础。

（2）软件过程的改进需要全体有关人员的积极参与，他们不仅需要有改进的热情和明确的目标，而且需要对当前过程有很好的了解。

（3）任何过程改进都有一定的风险，都有一个实践、改进、评审直至完善的循环往复、持续改善的过程，不可能一蹴而就。

（4）项目组的开发人员需要经过 PSP 的培训，使之具备自我改进的能力。

（5）整个开发单位的能力成熟度在总体上应处于 CMM 模型 2 级以上。

1.8.2　TSP 设计和实施原则

在软件开发（或维护）的过程中，首先需要按照 TSP 框架定义一个过程。在设计 TSP 过程时，需要遵循以下 7 条原则：

（1）循序渐进的原则。首先在 PSP 的基础上提出一个简单的过程框架，然后逐步完善。

（2）迭代开发的原则。选用增量式迭代开发方法，通过几个循环开发一个产品。

（3）质量优先的原则。对按 TSP 开发的软件产品，建立质量和性能的度量标准。

（4）目标明确的原则。对实施 TSP 的群组及其成员的工作效果提供准确的度量。

（5）定期评审的原则。在 TSP 的实施过程中，对角色和群组进行定期的评价。

（6）过程规范的原则。对每一个项目的 TSP 规定明确的过程规范。

（7）指令明确的原则。对实施 TSP 中可能遇到的问题提供解决问题的指南。

在实施 TSP 的过程中，应该自始至终贯彻集体管理与自我管理相结合的原则。具体地说，应该实施以下 6 项原则：

（1）计划工作的原则。在每一阶段开始时要制订工作计划，规定明确的目标。

（2）实事求是的原则。目标不应过高也不应过低，而应实事求是。在检查计划时，如果发现未能完成或已经超越规定的目标，应分析原因，并根据实际情况对原有计划做必要的修改。

（3）动态监控的原则。一方面应定期追踪项目进展状态并向有关人员汇报，另一方面应经常评审自己是否按 PSP 原理进行工作。

（4）自我管理的原则。团队成员如发现过程不合适，应主动、及时地进行改进，以

保证始终用高质量的过程来开发高质量的软件，任何消极埋怨或坐视等待的态度都是不对的。

（5）集体管理的原则。项目开发小组的全体成员都要积极参加和关心小组的工作规划、进展追踪和决策制订等项工作。

（6）独立负责的原则。按 TSP 原理进行管理，每个成员都要担任一个角色。在 TSP 的实践过程中，TSP 的创始人 Humphrey 建议在一个软件开发小组内把管理的角色分成客户界面、设计方案、实现技术、工作规划、软件过程、产品质量、工程支持以及产品测试 8 类。如果小组成员的数目较少，则可将其中的某些角色合并；如果小组成员的数目较多，则可将其中的某些角色拆分。总之，每个成员都要独立相当一个用色。

1.8.3 TSP 的度量

软件开发团队按 TSP 进行开发、维护软件或提供服务，其质量可用两组元素来表达。一组元素用以度量开发团队的素质，称为开发团队素质度量元；另一组用以度量软件过程的质量，称为软件过程质量度量元。

开发团队素质度量元主要有以下 5 个方面：所编文档的页数；所编代码的行数；花费在各开发阶段或各开发任务上的时间（以分钟为度量单位）；在各个开发阶段中引入和改正的差错数目；在各个阶段对最终产品增加的价值。应该指出，这 5 个度量元是针对软件产品的开发来陈述的，对软件产品的维护或提供其他服务，可以参照这些条款给出类似的陈述。

软件过程质量度量元有以下 5 项：设计工作量应大于编码工作量；设计评审工作量至少应占一半以上的设计工作量；代码评审工作量应占一半以上的代码编制的工作量；每千行源程序在编译阶段发现的差错不应超过 10 个；每千行源程序在测试阶段发现的差错不应超过 5 个。

无论是开发团队的素质，还是软件过程的质量，都可用一个等五边形来表示，其中每一个基本度量元是该等五边形的一个项。基本度量元的实际度量结果，落在其顶点与等五边形中心的连线上，其取值可以根据事先给出的定义来确定。在应用 TSP 时，通过对必要数据的收集，项目组在进入集成和系统测试之前能够初步确定模块的质量。如果发现某些模块的质量较差，就应对该模块进行精心的复测，有时甚至有必要对质量特别差的模块重新进行开发，以保证生产出高质量的产品，且能节省大量的测试和维护时间。

1.8.4 TSP 的流程

TSP 一般将一个软件项目的开发工作分为 4 个阶段，分别是需求阶段、设计阶段、实现阶段、集成阶段。任何一个应用 TSP 的项目可以只包括其中的一个阶段，也可以包括几个连续的阶段。

在项目开始之前，项目组应该执行启动过程，对整个任务进行全面地规划和组织。

在每个阶段之前，项目组应该执行重启过程，对下一个阶段的任务进行规划。

一般来说，如果项目组的成员经过了 PSP 的培训，项目组的启动过程约需 3 天时间，重启过程约需 2 天时间。此时，项目组与管理人员一起评审项目计划和分析关键风险。在项目已经启动之后，项目组应每周进行一次项目进展讨论会，另外还应及时向有关主管和客户报告项目的进展情况。

TSP 使用 23 个过程指南、14 个数据表格和 3 个标准。在这些过程指南中定义了 173 个启动和开发步骤。每一个步骤都不复杂，但它们的描述都非常详细，以便开发人员能够清楚地知道下一步应该做什么，应该怎样去做。这些过程指南可用来指导项目组来完成启动过程和一步步地完成整个项目。

经过 3 天的项目启动过程之后，项目组应该产生以下结果：项目组的目标；项目组各成员的明确角色；过程开发计划；项目组的质量计划；全面的开发计划和进度计划；下一阶段每个成员的详细工作计划；项目的风险分析结果以及项目的状态报告。

1.9　CMM/TSP/PSP 三者的结合

CMM/CMMI、PSP 和 TSP 为软件产业提供了一个集成化的、三维的软件过程改进框架。要特别注意的是，如果一个组织单纯实施 CMM/CMMI，永远不能真正做到能力成熟度的升级，而需要将实施 CMM/CMMI 与实施 PSP 和实施 TSP 有机地结合起来，才能达到软件过程持续改善的效果。

CMM/CMMI、PSP 和 TSP 组成的软件过程框架如图 1-6 所示。

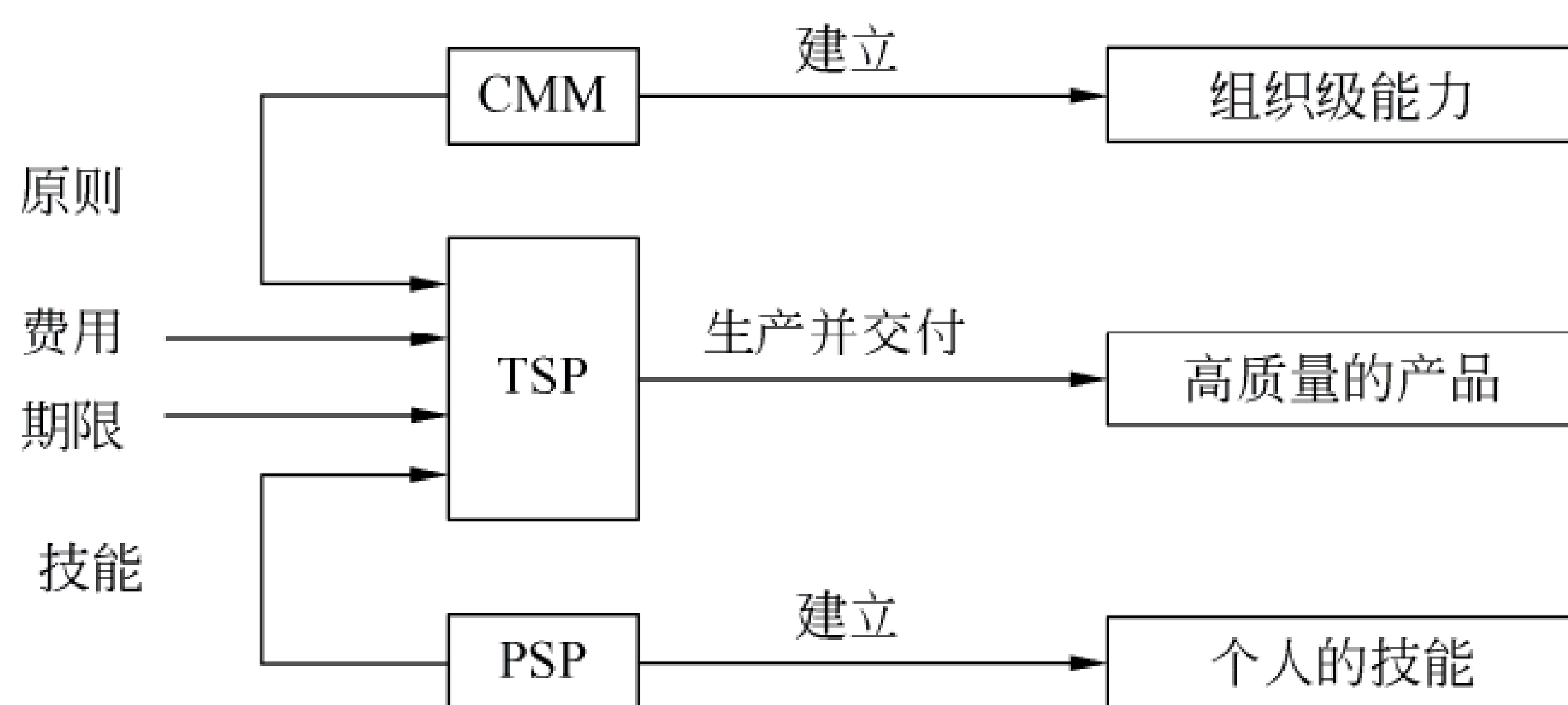


图 1-6　软件过程框架图

从图 1-6 可以看出，CMM/CMMI 是过程改善的第一步，它提供了评价组织的能力的方式，并为 TSP 提供了指导原则。组织只有开始实施 CMM/CMMI 后，才能认识到质量的重要性，注重对员工进行培训，合理分配项目人员。PSP 能够指导软件工程师如何保证自己的工作质量，估计和规划自身的工作，度量和追踪个人的表现，管理自身的软件过程和产品质量。PSP 为 TSP 的实施提供了软件工程师的个人技能。TSP 结合了

CMM/CMMI 的管理方法和 PSP 的工程技能，通过告诉软件工程师如何将 PSP 结合进 TSP，通过告诉管理层如何支持和授权项目小组，向组织展示如何应用 CMM/CMMI 的原则和 PSP 的技能去生产高质量的产品。

CMM 的 18 个关键过程域与 PSP 和 TSP 的对应关系如表 1-3 所示。

表 1-3　CMM 的关键过程域与 PSP 和 TSP 的对应关系

级　别	CMM 的 18 个关键过程域	提　供　者
优化级	缺陷预防	PSP
	技术变更管理	PSP
	过程变更管理	PSP
可管理级	定量的过程管理	PSP
	软件质量管理	PSP
已定义级	组织过程焦点	PSP
	组织过程定义	PSP
	培训大纲	无
	集成软件管理	PSP
	软件产品工程	PSP
	组织协调	TSP
	同行专家评审	PSP
可重复级	需求管理	TSP
	软件项目规划	PSP
	软件项目追踪和监控	PSP
	软件子合同管理	无
	软件质量保证	TSP
	软件配置管理	TSP

本章参考文献

[1] CMMI Product Development Team. Appraisal Requirements for CMMISM，Version 1.1：Technical Report. CMU/SEI-2001-TR-034

[2] 周伯生，吴超英，任爱华. CMMI 精粹——集成化过程改进实用导论. 北京：机械工业出版社，2002

[3] 张友生. 软件企业如何实施 CMM. http://www.csai.cn\pubcmm\howcmm.htm

[4] 张友生. 个体软件过程的改进. http://www.csai.cn\pubcmm\howpsp.htm

[5] 周伯生. 群组软件开发过程. http://www.csai.cn\pubcmm\tsp.htm

[6] 王瑛珑. 企业质量管理规范. 中国系统分析员.2003（5）：39-58

第 2 章　J2EE 与.NET 平台

以 SUN 为首的 Java 联盟 J2EE 平台和 Microsoft 推出的.NET 平台都是经常应用于企业级应用开发的平台。从这两个企业级应用开发平台一推出，业界就没有停止过对比和争论。究竟哪一个更适合企业的应用需求呢？本章将介绍 J2EE 和.NET 的平台特点，让读者在面对具体需求时，可以选择合适的解决方案。

2.1　J2EE 平台概述

自 Java 面世以后，Java 虚拟机（Java Virtual Machine，JVM）平台无关性的特点吸引了众多技术人员和厂商。在此之前，开发语言受限于运行的环境，例如，一个 Visual Basic 的程序员无法开发 UNIX 的应用程序，基于 DOS/Windows 的 C/C++开发的程序也不能运行在 Linux 中。而 Java 解决了这个问题。同样是掌握一种技术，掌握了 Java 则有更广阔的应用环境，技术员和软件厂商当然更愿意选择 Java。因此，Java 技术得到了快速的发展。

也就是在这个时候，企业应用进入了快速增长期。随着计算机硬件价格的不断下降、网络技术的快速发展、技术人员越来越多，开发、使用信息系统的成本快速降低。信息技术（Information Technology，IT）系统在更多的企业、更多的领域得到应用，企业应用的需求变得更加复杂，基于网络的分布式应用成为企业应用的标准模式。瘦客户端、多层计算的概念变得流行起来。

在今天，多层计算的概念已经深入人心，几乎每一位技术人员都知道需要将系统分层开发。但在 J2EE 刚刚推出的年代，多层计算还是阳春白雪。只有基于微软平台的分布式构件对象模型（Distributed Component Object Model，COM/DCOM）和对象管理集团（Object Management Group，OMG）发布的公共对象请求代理架构（Common Object Request Broker Architecture，COBRA）。COM/DCOM 是微软提供的解决方案，不能实现跨平台的应用，而 COBRA 是一套大而全的标准，学习、掌握 COBRA 都不是一件容易的事情。在这种背景下，完全基于 Java 的 Java2 平台企业版（Java 2 Platform Enterprise Edition，J2EE）标准推出，为开发分布式应用提供了简单易用的解决方案。

2.1.1　分布式的多层应用程序

J2EE 平台采用了多层分布式应用程序模型，实现不同逻辑功能的应用程序被封装到不同的构件中，处于不同层次的构件被分别部署到不同的机器中。图 2-1 表示了两个多

层的 J2EE 应用程序，根据下面的描述被分为不同的层。其中涉及的 J2EE 应用程序的各个部分将在 2.1.2 节中给出详细描述。

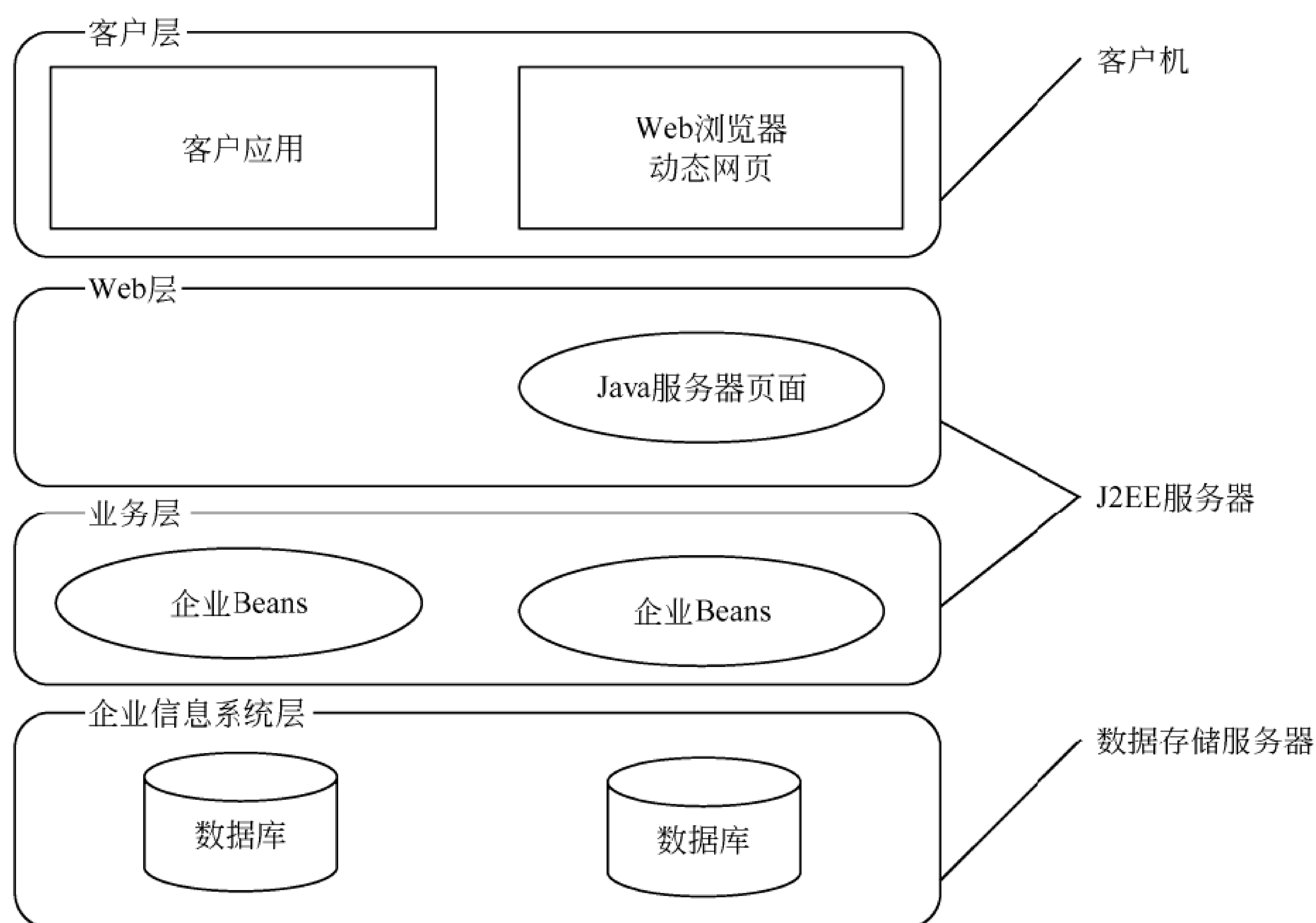


图 2-1　多层结构的应用程序

从图 2-1 中可以看到，J2EE 应用程序本身已经属于多层模型。一般来说，J2EE 应用程序经常分布于三个不同的位置，所以通常将 J2EE 应用程序的多层结构考虑为三层结构。这三个位置分别是客户端机器、J2EE 服务器和在后端数据存储服务器。三层结构的应用程序可以理解为在标准的两层结构中的客户端程序和后端服务中间增加了应用服务器。

应用服务器可以解决传统的两层客户/服务器计算中的不足，并且能够提供许多新的优点：

（1）可升级性。在传统的两层计算模式下，工作的服务器只能够有一台，而无论花费多少钱购买最先进的超级服务器，其计算能力也是有限的。而采用了应用服务器之后，可以利用负载均衡技术将计算工作量在几台服务器之间进行分担，因此，计算能力的提高可以通过增加中等规模的服务器来实现，以量取胜。而且，在原来的模式下，要更新设备时，不得不停机，将对业务造成影响，而采用了应用服务器后，在不影响原有系统工作的前提下，可以直接将新服务器安装部署起来，分担计算压力。

（2）分布式处理。采用应用服务器，数据库和应用服务器可以尽可能地按照靠近要完成工作的地方部署，从而最大可能地降低网络传输量。另外，分布式还可用于将远程

数据进行本地化存储。

（3）可重用的业务对象。应用服务器是一个反映业务处理过程的服务和对象的仓库，一旦应用服务器开发、实现完成后，其中的对象和服务就可能为另一个应用所重用。而且由于其都拥有标准接口和构件模型，因此重用更加容易而且能够降低成本。

（4）业务规则。在两层计算模式中，其软件设计主要强调以数据为中心，因此，对业务规则的处理显得有些不够有力。而在应用服务器的开发中，则强调业务对象的构建，很容易在计算机系统中封装模拟实际的业务规则和处理过程。

（5）跨平台集成。应用服务器在跨平台集成方面已经做了大量的基础工作，因此，开发人员不必关心底层的数据格式转换、字节顺序等与平台相关的因素，使得跨平台集成、跨平台部署更容易实现。

2.1.2　J2EE 构件

J2EE 应用程序由一系列的构件组合而成。一个 J2EE 构件就是一个软件单元，它随同与它相关的类和文件被装配到 J2EE 应用中，并与其他构件通信。J2EE 构件由 Java 语言编写而成，并和用 Java 语言编写的其他程序一样进行编译。J2EE 构件和“标准的”Java 类的不同点在于：它被装配在一个 J2EE 应用中，具有固定的格式并遵守 J2EE 规范，由 J2EE 服务器对其进行管理。J2EE 规范是这样定义 J2EE 构件的：客户端应用程序和 applet 是运行在客户端的构件；Java Servlet 和 Java Server Pages（Java 服务器页面，JSP）是运行在服务器端的 Web 构件；企业 Java Bean（Enterprise Java Bean，EJB）构件是运行在服务器端的商业构件。

1．J2EE 客户端

图 2-2 显示了客户层组成的多种方式。客户层可以直接和运行在 J2EE 服务器中的业务层通信，也可以通过运行在 Web 层中的 JSP 页面和 Servlet 与商业层构件进行通信。J2EE 客户层可以分为 Web 客户端、Applets 和 Java 应用。

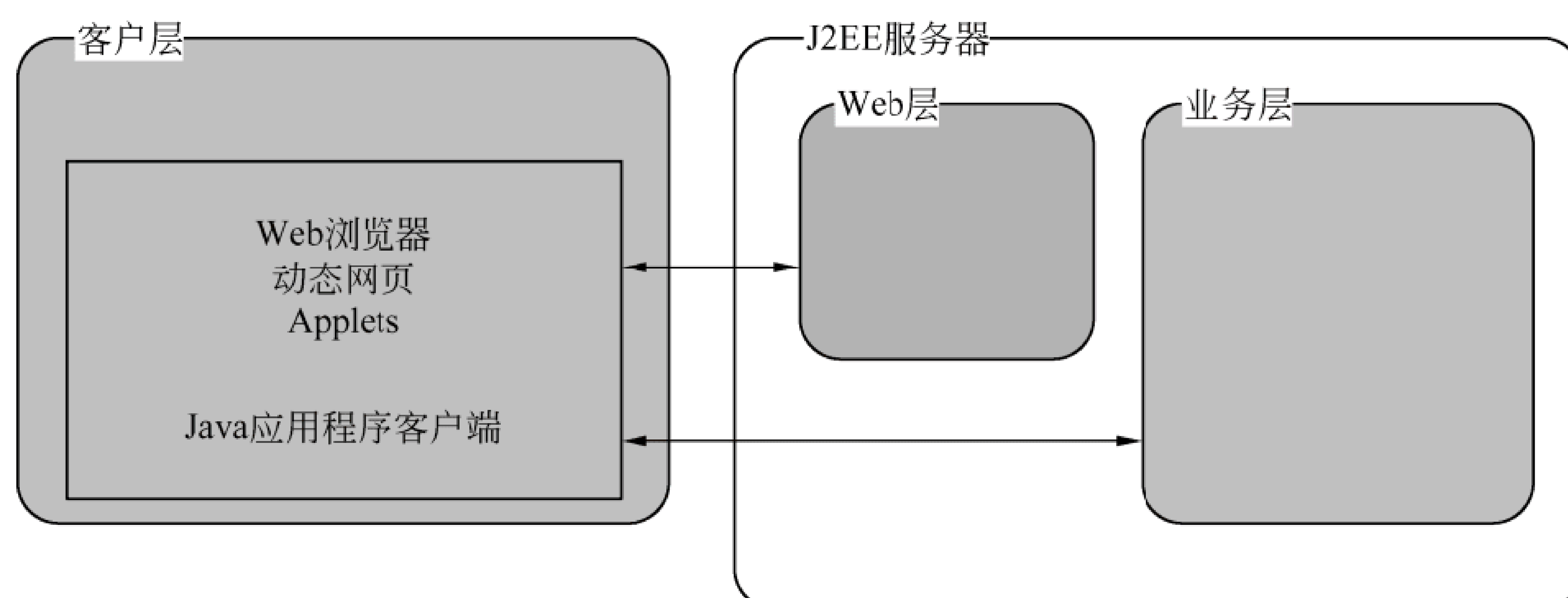


图 2-2　服务器通信

（1）Web 浏览器。Web 客户端也被称为瘦客户端，一般由各种浏览器承担，在浏览器中显示由 JSP 或 Servlet 动态生成的 Web 页面。在瘦客户端中的 Web 页面不会直接进行数据库查询或执行业务逻辑，而仅仅是向 Web 服务器提交处理请求，并显示 Web 服务器的处理结果。在 J2EE 架构中，使用瘦客户端时，重量级的操作都被交给了在 J2EE 服务器执行的 EJB。这样可以充分发挥 J2EE 服务器在安全性、速度、耐用性和可靠性方面的优势。有关这方面的详细知识，将在第 11 章进行介绍。

（2）Java Applet。Java Applet 也可以用于连接 J2EE 应用。Java Applet 是一种特定的 Java 程序，Java Applet 最大的特点就是它是在本地浏览器中执行。当浏览嵌有 Applet 的 Web 页面时，浏览器会将 Applet 程序同 Web 页面一起下载到本地计算机并使用位于本地计算机的 JVM 解释执行。因此，Applet 同时拥有类似于纯浏览器/服务器（Browser/Server，B/S）结构的易于安装、分发、维护等优点，同时也可以直接使用 Java 语言进行开发，使用 Java Swing 等图形用户界面（Graphical User Interface，GUI）的优点。

Applet 可以直接通过远程方法调用/互联网内部对象请求代理协议（Remote Method Invocation/Internet Inter-ORB Protocol，RMI/IIOP）等方式连接位于应用服务器的 EJB，从而达到表现层与业务逻辑层相分离的目的，实现三层架构的系统。

不过，使用 Java Applet 作为 EJB 的客户端更像在 Browser + JSP 方案与 Java 应用程序方案的折衷。虽然 Java Applet 具有像 Web 应用一样的易于分发、管理、维护的优点，但效率和兼容性稍差。在浏览器中运行的 Java Applet 通常速度很慢，且对于浏览器的版本和本地 JVM 的版本存在不兼容的可能。而 Java Applet 具有 Java 应用程序的一些优点，比如说可以直接使用 Java Swing 开发复杂的 GUI，但其实 Java Applet 技术很难直接满足用户对客户端复杂的要求，例如多文档界面（Multiple Document Interface，MDI）风格的应用程序。

（3）Java 应用程序。Java 应用程序是运行在客户端机器中的本地应用程序，由本地的 JVM 负责解释执行。毋庸置疑，通过 AWT 和 Swing，Java 应用程序可以提供相对 Web 更丰富强大的应用界面，并且能够更快捷的响应本地事件。不过，大规模的维护与分发对于 Java 应用程序而言是比较困难的，这也限制了 Java 应用程序的应用范围。

Java 应用程序可以直接访问运行在业务层的 EJB，也可以通过超文本传输协议（Hypertext Transfer Protocol，HTTP）访问运行在 Web 容器中的 Servlet，并通过 Servlet 达到业务处理的目的。

2．J2EE 中间层

J2EE 中间层的内容极为丰富，也是 J2EE 架构的核心，绝大多数的 J2EE 应用程序都会将业务逻辑部署在中间层，EJB 是 J2EE 中间层中最重要也是最有特点的构件之一。图 2-3 显示了一个 EJB 如何从客户端接受数据，对它进行处理（如果需要），并将其发送到数据库进行存储。EJB 也可以从持久化的存储设备中获取数据，对它进行处理（如果需要），并将其发送到客户端应用程序。

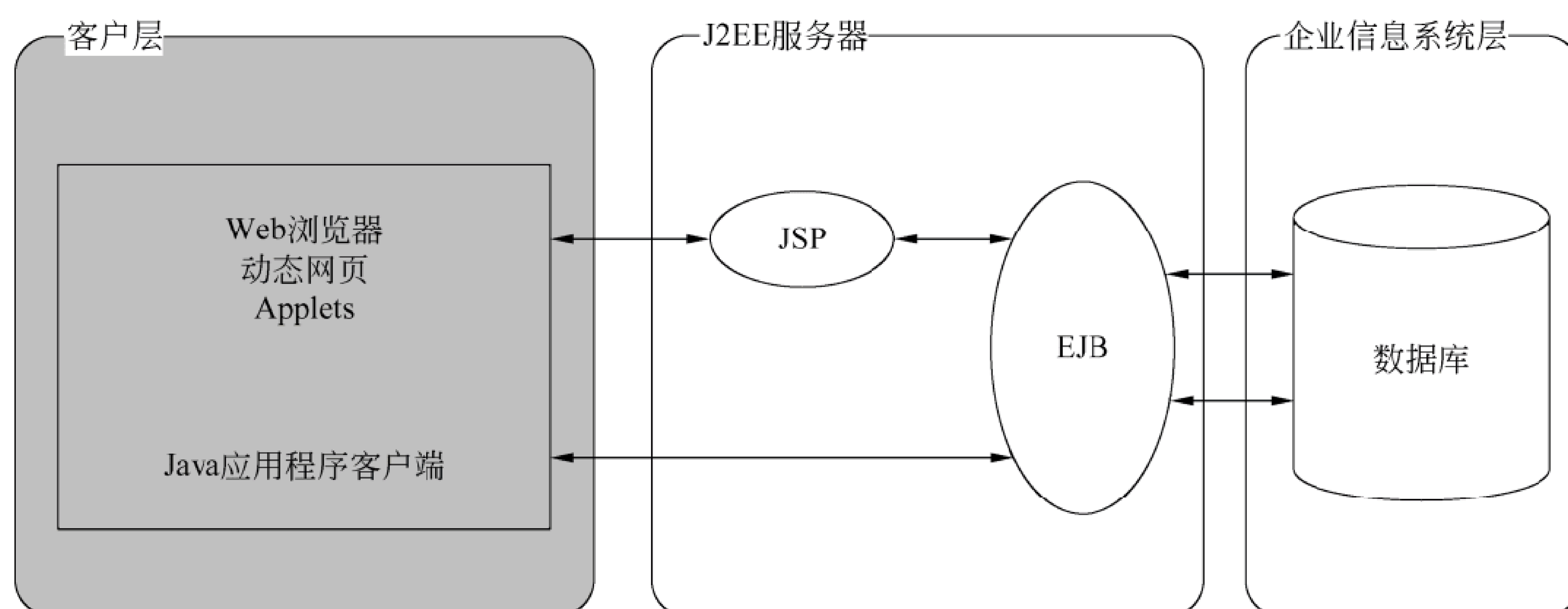


图 2-3　业务层和 EIS 层

SUN 在 EJB2.0 规范中对 EJB 定义如下：EJB 是用于开发和部署多层结构的、分布式的、面向对象的 Java 应用系统的跨平台的构建体系结构。使用 EJB 编写的应用程序具有可扩展性、交互性以及多用户安全的特性。这些应用只需要写一次，就可以发布到任何支持 EJB 规范的服务器平台上。

从名字上看 EJB 似乎仅仅是 Java Beans 的企业版，但 EJB 绝对不是 Java Beans 的简单升级。最初的 Java Beans 是 Java 语言中的一种构件模型，其重点是允许开发人员可以在开发工具中可视化的操作构件，Java Beans 可以被集成到任何 Java 程序中，尤其是在 Java Applet 和应用程序中得到了大量的应用。而 EJB 是一种非可视化构件，EJB 完全运行于服务器端。EJB 可以和远程程序通信，并提供响应的功能。如果 EJB 不和客户端程序交互，则不执行具体的功能。同 Java Beans 不一样的是，EJB 仅仅在网络计算的环境下才有意义。

EJB 可以分为三种类型，分别是会话 Bean（Session Beans，SB）、实体 Bean（Entity Beans，EB）和消息驱动 Bean（Message-driven Beans，MB）。一个会话 Bean 描述了与客户端的一个短暂的会话。当客户端的执行完成后，会话 Bean 和它的数据都将消失；实体 Bean 对应数据实体，它描述了存储在数据库的表中的持久数据。如果客户端终止或服务结束，底层的服务会负责实体 Bean 数据的持久性（也就是将其存储到某个地方，如数据库）；消息驱动 bean 是 EJB2.0 新增的类型，它结合了一个会话 Bean 和一个 Java 信息服务（Java Messaging Service，JMS）信息监听者的功能。客户把消息发给 JMS 目的地，然后 JMS 提供者和 EJB 容器协作，把消息发送给消息驱动 Bean。

3．企业信息系统层

企业信息系统层处理企业信息系统软件，并包含诸如企业资源计划、主机事务处理、数据库系统和其他传统系统等一些底层系统。J2EE 应用程序构件可能需要访问企业信息系统。J2EE1.3 支持连接件（Connector）架构，该架构是将 J2EE 平台连接到企业信息系

统上的一个标准应用程序接口（Application Programming Interface，API）。

2.1.3 J2EE 容器

如果把各种各样的构件理解为一大堆水果的话，容器就是用来存放这些水果的箱子。在 J2EE 水果族中有很多构件，例如类似于苹果的 Servlet 构件和类似于桔子的 EJB 等。如果不对这些水果进行管理，扔的到处都是，那么会出现一系列的情况，例如：

（1）找不到想要的苹果或桔子，要么找到的是一些没有成熟的水果，要么甚至可能会取错水果。

（2）缺乏保鲜处理的水果会很快的腐烂掉。

（3）其他问题。

很明显，需要管理起这些水果，可以在需要的时候放心的享用它。于是把这些水果分门别类的放到箱子里面，在上面贴上标签，水果箱可以保持水果长久的新鲜，并在需要的时候智能地从箱子的某个角落里翻出想要的水果。如前所述，这个存放 J2EE 水果的、无所不能的箱子就是 J2EE 容器。

要知道，如果从零开始，多层应用程序是很难编写的。开发者需要花费大量的精力来完成事务处理、状态管理、多线程、资源池和其他底层处理。于是，J2EE 容器为 J2EE 标准中每一个构件类型提供底层服务，用户完全不需要自己开发这些服务，所以可以全力以赴地着手解决商业问题。

在容器中可包含若干构件，并为这些构件提供服务。Web 构件、EJB 等都必须首先被装配到一个 J2EE 应用程序中，并且部署到相应的容器后才可以执行。部署时会将 J2EE 应用程序构件安装到 J2EE 容器中，如图 2-4 所示。

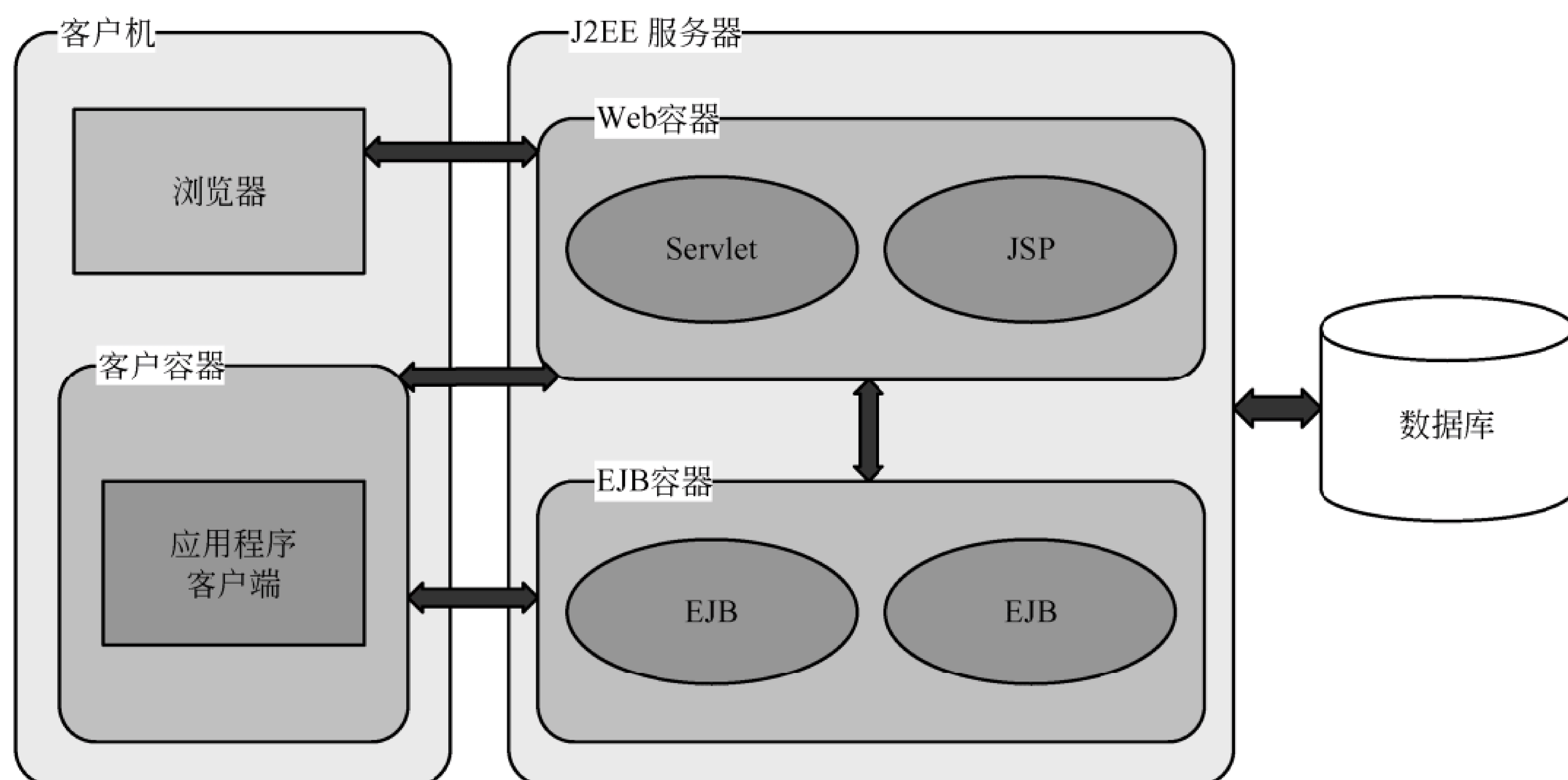


图 2-4 J2EE 服务器和容器

（1）J2EE 服务器：J2EE 服务器是 J2EE 产品的运行容器。一个 J2EE 服务器提供 EJB 容器和 Web 容器。

（2）EJB 容器：管理它所包含的 EJB，容器负责对象的注册、提供远程接口、创建和清除对象实例、检查对象安全性、管理对象的活动并协调分布式事务处理。

（3）Web 容器：管理 JSP 页面和 Servlet 构件的执行。Web 构件和 Web 容器运行在 J2EE 服务器中。

（4）客户端应用程序容器：管理应用程序客户端构件的运行。应用程序客户端和它的容器运行在客户端中。

2.1.4　J2EE 的部署

J2EE 构件被分别打包并绑定到一个 J2EE 应用中以供部署。每一个构件、构件相关资源，例如 GIF、超文本标记语言（HyperText Mark-up Language，HTML）文件，和一个部署说明组成了一个模块并被添加到 J2EE 应用程序中。一个 J2EE 应用由一个或几个 EJB 构件、Web 构件或应用程序客户端组成。根据不同的设计需求，最终的企业解决方案可以是一个 J2EE 应用程序，也可以由多个 J2EE 应用程序组成。

一个 J2EE 应用程序以及它的每一个模块有它自己的部署说明。一个部署说明就是一个可扩展标记语言（eXtensible Markup Language，XML）文件，它描述了一个构件的部署设置。例如，它可以描述一个 EJB 事务属性和安全性授权。部署说明信息是公开的，改变部署说明不必修改源代码。在运行时，J2EE 服务器将按照部署说明中的描述执行 J2EE 应用。

一个 J2EE 应用以及它的所有模块被提交到一个企业文档（Enterprise Archive，EAR）文件中。一个 EAR 文件就是一个具有.ear 扩展名的标准的 Java 文档（Java Archive，JAR）文件。在 J2EE 软件开发包（Software Development Kit，SDK）中有程序部署工具的 GUI 版本。通过这个部署工具可以建立 EAR 文件，并在其中添加 JAR 文件和 Web 文档（Web Archive，WAR）文件。

（1）每一个 EJB JAR 文件包含一份部署说明、一组 EJB 以及相关的文件。

（2）每一个应用程序客户端的 JAR 文件包含一份部署说明、应用程序客户端的类文件以及相关的文件。

（3）每一个 WAR 文件包含一份部署说明、Web 构件以及相关的资源。

使用模块和 EAR 文件可以很方便地使用同一构件装配出不同的 J2EE 应用。不需要额外的编程工作，开发人员唯一要做的就是在 J2EE EAR 文件中添加各种 J2EE 模块。

2.1.5　Java EE

2006 年 5 月 11 日，千呼万唤始出来的 Java EE 5 终于最终完成。为了体现出 Java EE 5 中做出的巨大的变化，Java EE 5 一反以往的 J2EE 1.x 的命名方法，将本应命名为 J2EE

1.5 的最新版本命名为 Java EE 5。Java EE 5 从很多开源项目中吸取了不少养分，更关注开发与部署的快捷简便，增加了对轻量级容器的支持、Java 服务器界面（Java Server Faces，JSF）等表现层技术、使用 Annotations 取代部署描述符、使用 EJB3.0 的标准简化 EJB 的开发、增强了对 Web Service 和面向服务的架构（Service-Oriented Architecture，SOA）的支持。Java EE 5 的架构如图 2-5 所示。

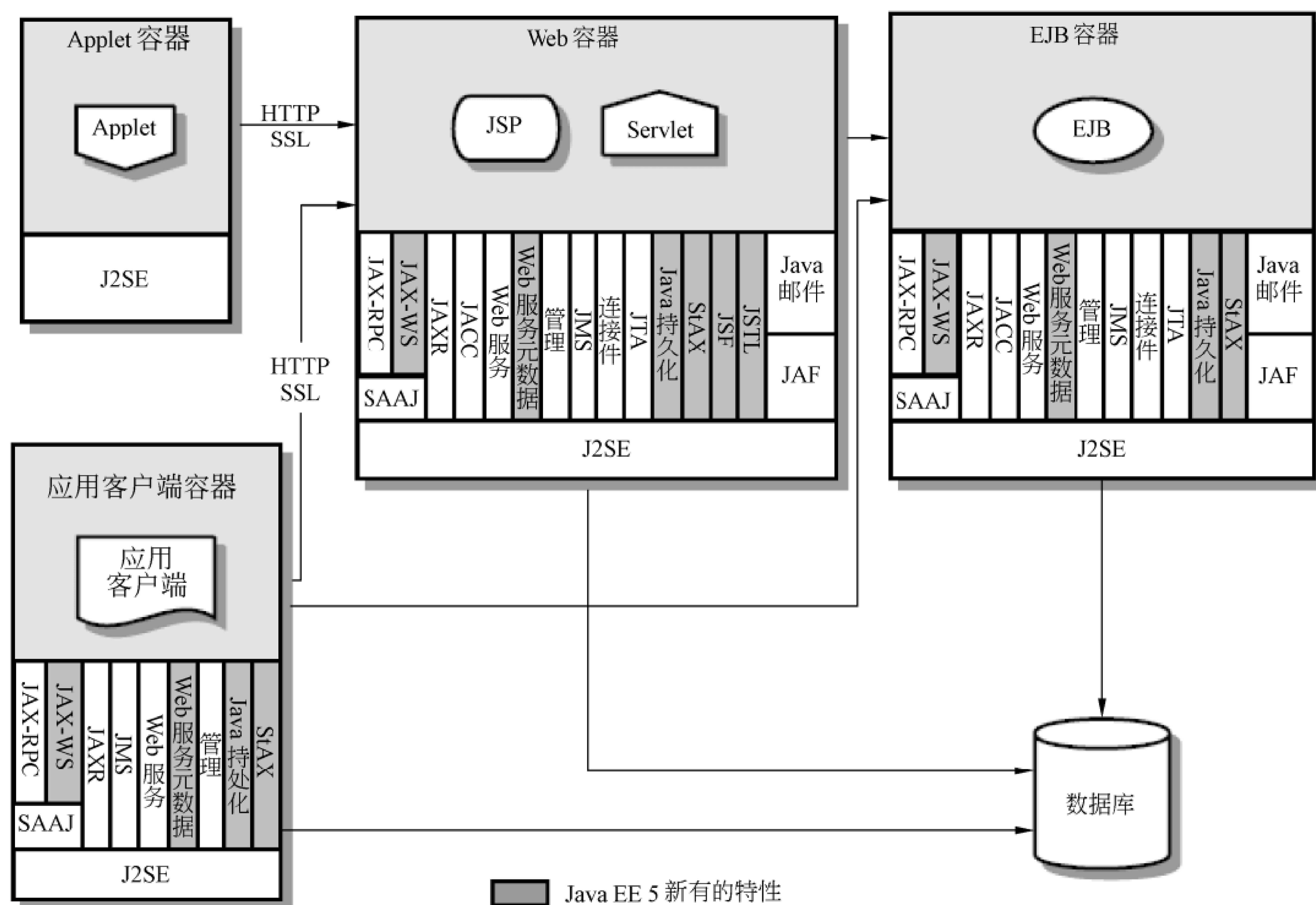


图 2-5　Java EE 5 架构

图 2-5 中灰色的部分为 Java EE 5 种新增的技术标准。从图 2-6 中可以看出，Java EE 5 与以往的 J2EE 1.X 是一脉相承的企业应用标准，是一个标准的多层应用模型。不过，与 J2EE 1.4 相比，无论是在 Web 容器还是 EJB 容器，都补充了更多的技术标准。例如，在增加了一系列用于 Web Service 和 SOA 的技术标准，补充了 Java Persistence 用于简单的 Java 对象（Plain Ordinary Java Objects，POJO）的持久化等等。根据 SUN 官方的说明，Java EE 5 的启动速度提高了 30%，而占用的内存下降了 30%，Web Service 的性能提高了 5 倍。

在 Java EE 5 中，SUN 推出了 EJB3.0 的标准，EJB3.0 与 EJB 的早期版本相比，有了很大的进步。恐怕大多数了解 EJB1.0 和 EJB2.0 的人都会说：“EJB 是个功能强大的东西”，不过马上又会补充一句：“就是太复杂了，学习、开发、部署都复杂”。的确，

SUN 创世纪般的推出了 EJB，甚至在很长的一段时间内，EJB 就是 J2EE 的代名词，J2EE 就是 EJB。但是 EJB 自出生开始，人们就没有停止对它的诟病。首先是 EJB 的性能，然后就是 EJB 的复杂。于是，在开源社区中，人们开始讨论一些轻量级容器，用以取代复杂的、重量级的 EJB 容器。由于轻量级容器具有学习、开发和部署相对简单的特点，虽然在企业级应用的支持上略显弱势，但毕竟大多数的应用并不需要为这些所谓的企业级特性上投入巨大的成本，因此轻量级容器得到了非常广泛的应用。在 EJB3.0 中大量吸收了这些轻量级容器的养分，将简化 EJB 开发作为首要目标，并引入了依赖注入（Dependency Injection）和新的对象关系映射（Object/Relation Mapping，ORM）持久化方案。这一切都让 EJB 变得更简单，更容易开发和使用。

2.2　.NET 平台概述

Microsoft（微软）公司在 2000 年 7 月发布了新的应用平台.NET，至今已经发展至.NET Framework 3.5。Microsoft .NET 平台包括 5 个部分，如图 2-6 所示。

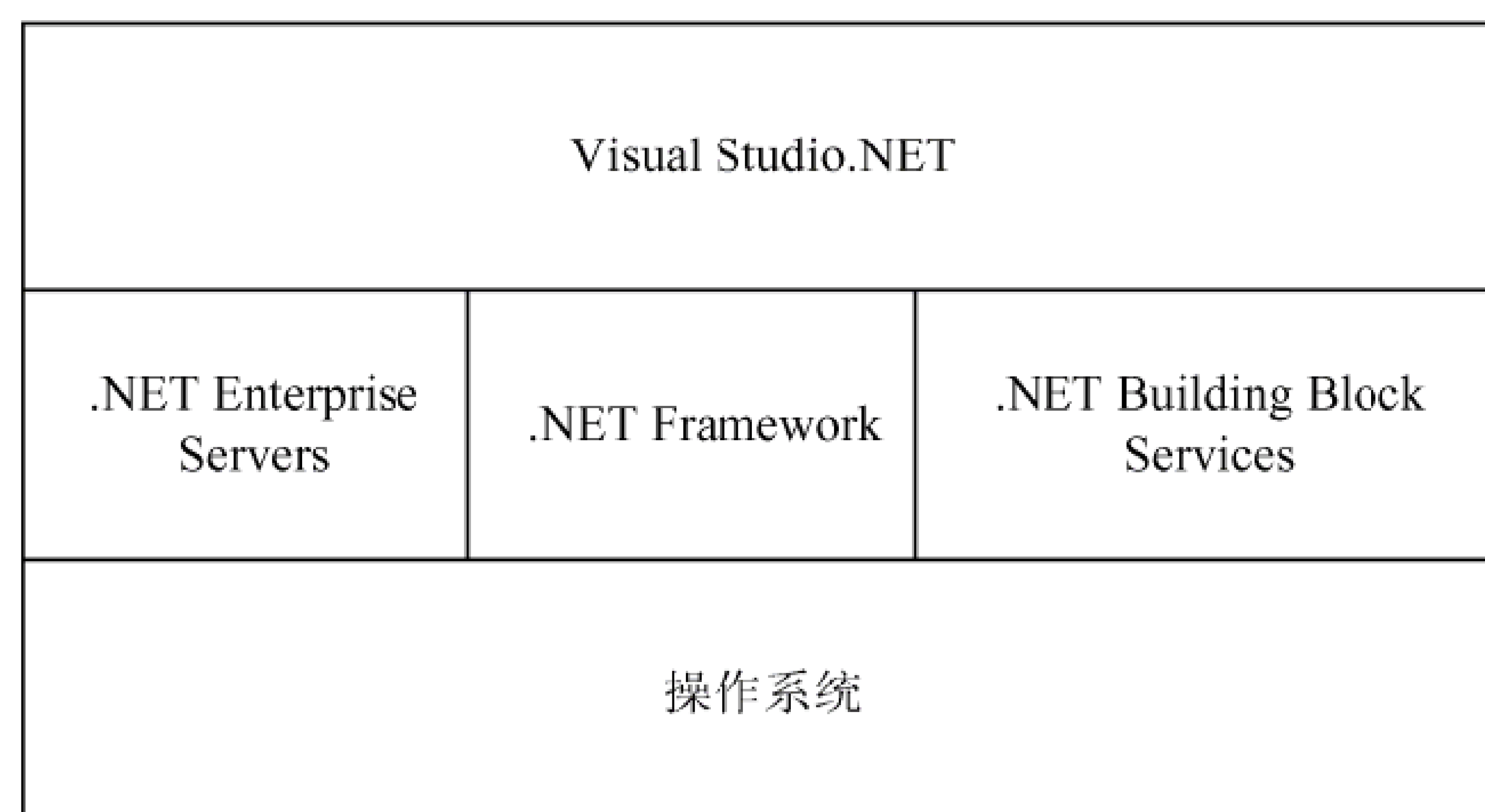


图 2-6　Microsoft .NET 平台

操作系统是.NET 平台的基础，在操作系统方面，Microsoft 有着强大的开发能力，目前的.NET 平台可以运行在包括 Windows 2008 Server 等多个 Microsoft 提供的操作系统中。

.NET Enterprise Servers 提供了一系列的.NET 服务器产品，包括 Application Center、BizTalk Server 和 Commerce Server 等。通过这些产品，可以缩短构建大型企业应用系统的周期。

.NET Building Block Services 是一些成型的服务，例如，由 Microsoft 提供的.NET Passport 服务等。.NET 的开发者可以以付费的方式直接将这些服务集成在自己的应用程序中。

.NET Framework 是整个.NET 平台的核心，.NET Framework 为开发.NET 应用提供了

底层的支持，并集成了一系列新的技术与构件，结合 Enterprise Server，为多层的分布式应用提供支持。

Visual Studio.NET 是.NET 应用程序的集成开发环境，它位于.NET 平台的顶端。Visual Studio.NET 是一个强大的开发工具集合，里面集成了一系列.NET 开发工具，如 C#.NET、VB.NET、XML Schema Editor 等。

2.2.1 .NET Framework

.NET Framework 是一整套的开发模型，图 2-7 给出了.NET Framework 1.1 的结构图。其中核心的部分就是通用语言运行时（Common Language Runtime，CLR）。CLR 是.NET 程序的执行引擎，.NET 的众多优点也是由 CLR 所赋予的。CLR 与 JVM 的功能类似，提供了单一的运行环境。任何.NET 应用程序都会被最终编译成为中间语言（Intermediate Language，IL），并在这个统一的环境中运行。也就是说，CLR 可以用于任何针对它的编程语言，这也就是.NET 的多语言支持。CLR 还负责.NET 应用程序的内存管理、对象生命期的管理、线程管理、安全等一系列的服务。将在 2.2.2 详细介绍 CLR。

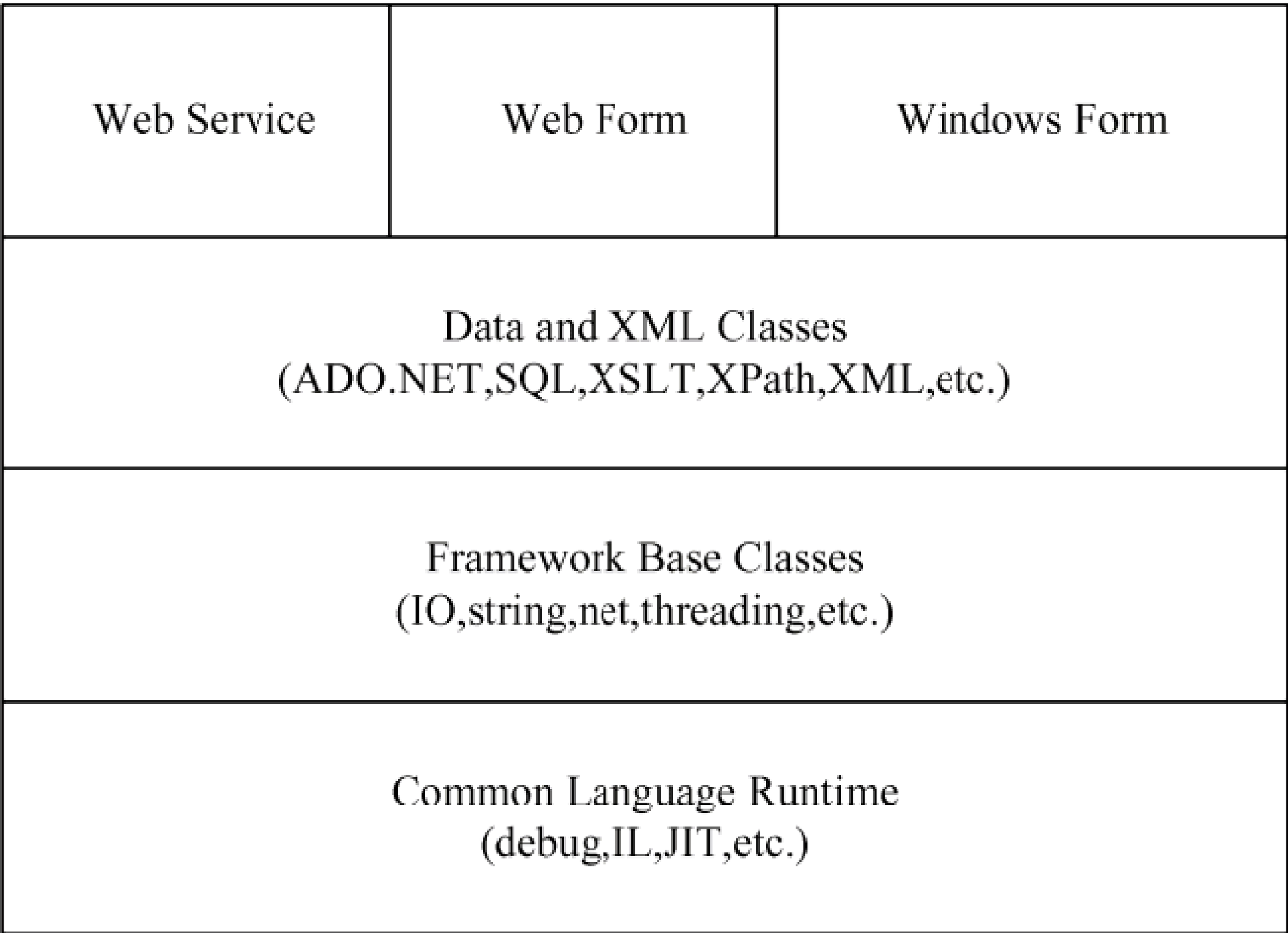


图 2-7 .NET Framework1.1 组成结构

.NET 为开发者提供了丰富的 API，每一种.NET 语言都可以使用这些 API。其中包含了大量的类供开发者使用，如图 2-7 所示。

跨平台是 Java 的一大特点，而.NET 可以认为是跨语言的。通过 CLR，使用某种.NET 语言开发的程序可以被其他的.NET 语言直接使用，从而充分利用各种语言的优点。例如，可以使用 VB.NET 书写 UI（User Interface，用户界面）相关的内容，而用 C++开发底层的计算功能。跨语言的特性对已经熟悉某种语言的程序员而言，进入.NET 会更加容易。

.NET Framework 的最新版本是 3.5，在.NET Framework 3.5 中，除了继续保持对

Windows GUI 程序、Web 应用、智能终端的支持外，Windows 通信基础（Windows Communiction Foundation，还增加了 WCF）、Windows 显示基础（Windows Presentation Foundation，WPF）、工作流（WorkFlow，WF）、语言级集成查询（Language INtegrated Query，LINQ）等一系列技术标准。应用.NET Framework 提供的 API，可以大大提升开发效率，减少开发成本。

1．WCF

顾名思义，WCF 适用于程序间相互通信的技术，在.NET Framework 3.0 中第一次出现，在 3.5 中进行了强化。WCF 是一个统一的，可用于建立安全，可靠的面向服务的应用高效的开发平台。WCF 是构建安全可靠的事务性服务的统一框架。它是一种构建分布式面向服务系统的非常丰富的技术基础，它统一了消息风格和远程过程调用（Remote Procedure Call，RPC）风格，并且通过二进制和基于开放标准的通信达到了平台最优化。在.NET Framework 刚刚推出的时候，微软即对 Web Service 提供了支持，在.NET Framework 2.0 中，微软推出了.NET Remoting 来解决 RPC 的问题。WCF 可以认为是对 Web Service、.NET Remoting 和 MSMQ 的整合，从而一举统一了分布式应用程序通信的问题。可以看出，WCF 不但是 Microsoft 公司 SOA 框架中的一个重要成员，在未来的微软云计算路线图中，可能也会占有一席之地。

2．WPF

WPF 是 Microsoft 公司用于 Windows 的统一显示子系统，它由显示引擎和托管代码框架组成，使开发人员和设计人员可以创建更好的视觉效果、不同的用户体验。

WPF 引擎统一了开发人员和设计人员体验文档、媒体和 UI 的方式，为基于浏览器的体验，基于窗体的应用程序、图形、视频、音频和文档提供了一个单一的运行时库。WPF 使得应用程序不仅能够充分利用现代计算机中现有的图形硬件的全部功能，而且能够利用硬件将来的进步。例如，WPF 的基于矢量的呈现引擎使应用程序可以灵活地利用高每英寸的点数（Dot Per Inch，DPI）监视器，而无需开发人员或用户进行额外的工作。同样，当 WPF 检测到支持硬件加速的视频卡时，它将利用硬件加速功能。

WPF 框架为媒体、用户界面设计和文档提供的解决方案远远超过开发人员现在所拥有的。WPF 的设计考虑了可扩展性，使开发人员可以完全在 WPF 引擎的基础上创建自己的控件，也可以通过对现有 WPF 控件进行再分类来创建自己的控件。WPF 框架的核心是用于形状、文档、图像、视频、动画、三维以及用于放置控件和内容的面板的一系列控件。这些“自有控件”为开发下一代用户体验提供了构造块。

Microsoft 公司在引入 WPF 的同时，还引入了可扩展应用程序标记语言（eXtensible Application Markup Language，XAML），这是一种公开表示 Windows 应用程序用户界面的标记语言，可使开发人员和设计人员用来构建和重用 UI 的工具更加丰富。对于 Web 开发人员，XAML 提供了熟悉的 UI 说明模式。XAML 还使 UI 设计从基础代码中分离出来，从而使开发人员和设计人员之间的合作更加紧密。

3．WF

WF 的核心是工作流引擎。工作流引擎可以应用于很多领域，在 J2EE 中，可以选择的工作流引擎非常多，而在.NET 中，工作流引擎一直是其软肋。与 WCF 一样，WF 也是在.NET Framework 3.0 中出现的。.NET Framework 支持两种工作流，分别是顺序工作流和状态机工作流，基本可以应付绝大多数需要使用工作流的需求。不但.NET 程序可以使用 API 访问、控制工作流，在 Visual Studio 2008 中，还集成了工作流定义的工具，让程序员可以以图形化的方式进行工作流程序的开发。有关工作流的更加详细的知识，请阅读本书第 16 章。

4．LINQ

目前，面向对象编程技术在工业领域的应用已经进入了一个稳定的发展阶段。程序员现在都已经认同像类、对象、方法这样的语言特性。考察现在和下一代的技术，一个新的编程技术的重大挑战开始呈现出来，即面向对象技术诞生以来，并没有解决降低访问和整合信息数据的复杂度的问题，其中两个最主要访问的数据源与数据库与 XML 相关。

LINQ 提供了一条更常规的途径，即给.NET Framework 添加一些可以应用于所有信息源的具有多种用途的语法查询特性，这是比向开发语言和运行时（runtime）添加一些关系数据特性或类似 XML 特性更好的方式。这些语法特性就叫做 LINQ。

5．BI

虽然商业智能（Business Intelligence，BI）本身不属于.NET Framework 的一部分，但是，在 Microsoft 公司的.NET 战略中，Visual Studio 和 SQL Server 为 BI 的开发提供了很好的支持。.NET 应用程序可以非常方便地集成 BI 应用程序，为企业应用提供 BI 解决方案。

在 SQL Server 2005 中，除去数据库引擎外，Microsoft 公司提供了 SQL Server 集成服务（SQL Server Integration Service，SSIS）、SQL Server 分析服务（SQL Server Analysis Service，SSAS）和 SQL Server 报表服务（SSRS）这三种服务。结合 SQL Server 数据库引擎和.NET 应用程序，可以构造完整的 BI 解决方案。

SSIS 的前身是 SQL Server 2000 中的 DTS，它是一个数据抽取、转换和加载（Extraction-Transformation-Loading，ETL）工具，可以方便地完成 ETL 任务；SSAS 与 SQL Server 2000 中的 Analysis Service 一样，它提供了对多维数据库的支持。SSAS 支持 XMLA 和多维表达式（Multi-Dimensional eXpression，MDX）查询，支持数据挖掘，可以很好的支持数据仓库；SSRS 可以连接关系数据库或 SSAS，能够方便的对报表进行开发和管理。在.NET 应用程序中，可以很好的集成 SSRS 中的报表，为.NET 应用程序提供强大的 BI 支持。

6．MONO

一般认为，Java 具有跨平台的优势，而.NET 应用程序只能运行在 Windows 环境中，

MONO 的出现改变了这个状况。

MONO 是一个开源项目，目前在 Novell 的主持下进行开发。MONO 的目标就是让.NET 可以运行在 Linux、FreeBSD、Unix、Mac OS 和 Solaris 环境下。通过 MONO，.NET 可以运行在非 Windows 平台下。目前 MONO 的最新版本是 2.2，其 2.0 版获得了 Developer.com 发布的 2009 年度.NET Tools 大奖。

希赛教育专家提示：在企业的实际应用中，使用 MONO 来实现.NET 跨平台的应用并不是一个好主意。在这些平台下，Java 会是更好的选择。

2.2.2　通用语言运行时

托管是.NET 中重要的概念，使用托管意味着代码可以被 CLR 所管理。这些程序则可以使用 CLR 提供的各种服务，例如，垃圾收集等。所以无论是用什么语言，只要采用了.NET 的托管机制，就能开发出具有垃圾自动收集、程序间相互访问等的.NET 框架应用程序。

1．程序集

可以这样理解程序集的概念：首先，程序集是一个或多个托管模块或资源文件的逻辑分组；其次，程序集是可重新使用、确保安全和版本控制的最小单元。图 2-8 是把托管模块组合称为程序集的示意图。

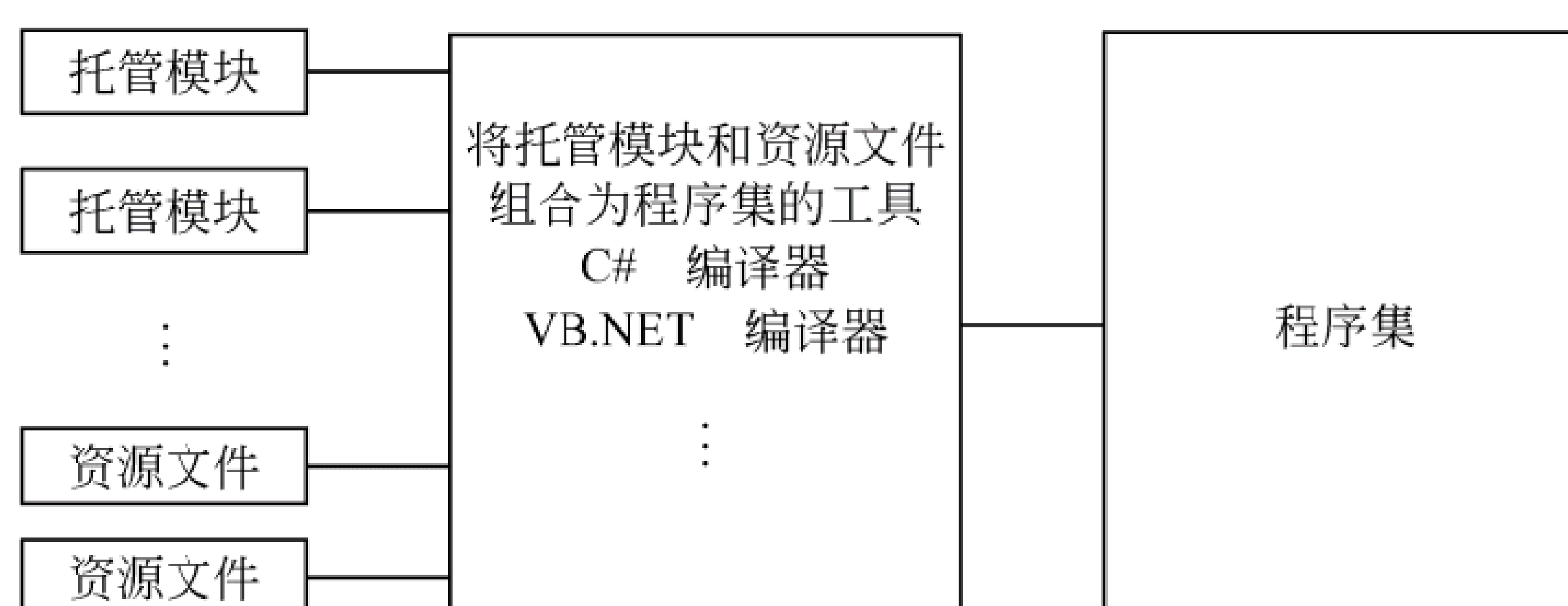


图 2-8　将托管模块组合为程序集

程序集允许用户将可重用、可部署的构件的逻辑部分和物理部分相分离。例如，可以将不常用的类型或资源放到独立的程序集文件中，这些独立的文件可以根据需要从 Web 站点上动态下载。如果不需要使用这些程序集，那么就永远不需要下载这些文件。

程序集中还包含被引用的程序集的自描述信息，例如，版本号等。CLR 可以通过这些自描述信息执行程序而不需要其他的附加信息。因此，程序集使得.NET 应用更容易部署。

2．中间语言

当托管代码被编译后，并不产生本机的二进制代码，而是产生了包含 IL 的程序集。

每种托管语言（如 C#，VB.NET 等）都产生 IL 程序集，IL 为被编译的代码提供一种通用的表示。Microsoft 公司宣称，编译为 IL 的代码可以运行在任何使用.NET Framework 的处理器（Central Process Unit，CPU）和操作系统中，从而为.NET 提供更多的应用平台。

被编译为 IL 的程序集不能够直接运行，CLR 中使用实时（Just In-Time，JIT）编译器将 IL 转化为本机的 CPU 指令。图 2-9 显示了首次调用一个方法的执行过程。

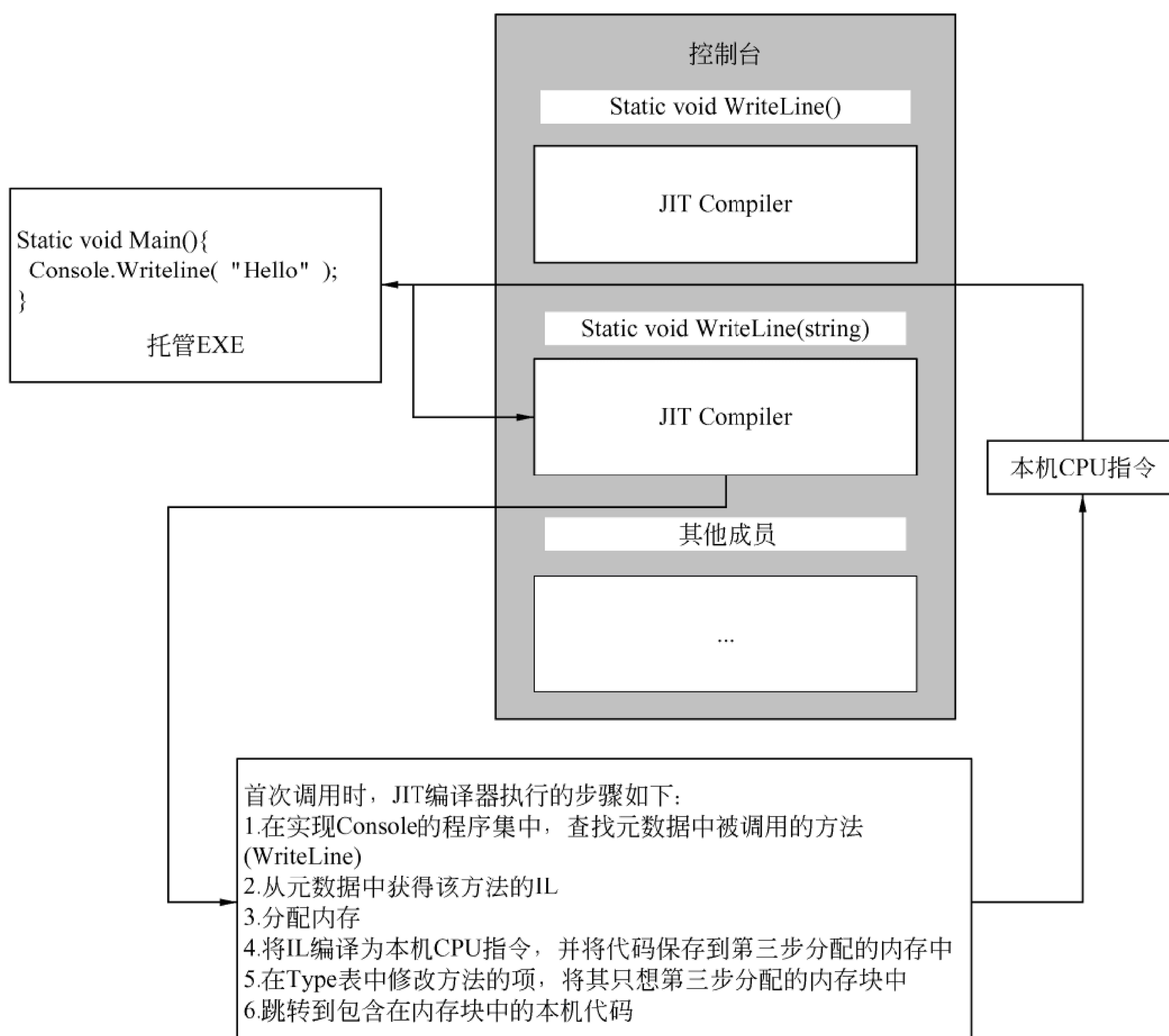


图 2-9 首次调用方法的执行过程

JIT 编译器负责将 IL 代码编译成本机的 CPU 指令，因为 JIT 是在程序第一次运行时实时的编译，所以 JIT 也称作实时编译器。只有当程序第一次运行时，JIT 才会工作，它会把编译好的本机指令存放在内存中，也就是说，程序在第一次运行的时候速度相对较慢，而以后的运行速度会加快很多。但是，如果程序终止，随着程序退出内存空间，编译好的本机指令也会随之退出内存，所以重新启动应用程序时会再一次启动编译过程。

如果程序的运行平台已经确定，同时对性能有较高的要求的话，.NET Framework 提供了一个小工具——NGen.exe，这个工具可以将所有程序集的 IL 代码编译成为本机代码并将编译好的本机代码保存在磁盘中。这样，CLR 可以在加载程序集时加载这个预编译的代码，从而提高运行速度。

2.3　J2EE 和.NET 平台的比较

首先，需要明确的是 J2EE 和.NET 的目标，这两个平台都是为了解决构建企业计算等大型平台而出现的。在这两个平台中都包含了一系列的技术，通过这些技术可以缩短开发周期，提高开发效率，节省构造成本，同时这两个平台都在安全性、扩展性、性能方面做出了努力，都提供了一系列的技术可供选择。从这个角度来说，这两个平台都实现了它们的目标，都是成功的。这两个平台要解决的问题类似，很多技术也非常类似，有些概念甚至仅仅是名称上的差别而已。本章对这两个平台进行对比并不是想说明这两个平台的哪一个更优秀，事实上，无论是 J2EE 还是.NET，都是优秀的平台解决方案。笔者仅仅是通过对比的手段加深读者对 J2EE 和.NET 技术的理解，能够在工作中根据实际需要确定选用的平台和技术，构造合理的解决方案。

2.3.1　JVM 与 CLR

JVM 是 J2EE 平台的底层支持，而 CLR 是.NET Framework 的核心。无论是 JVM 还是 CLR，都包含了许多新技术，对比这些技术的异同，有助于读者对这两个平台有更深的理解。

首先是字节码（bytecode）和 IL。JVM 为了提供平台无关性的支持，它将所有的代码都首先编译为 bytecode，然后，在运行过程中，JVM 对 bytecode 进行解释执行。IL 是.NET 提供的中间语言，所有的.NET 程序都将编译为 IL，在第一次运行时由 JIT 编译为本机代码，然后执行该代码。bytecode 和 IL 的机制本身没有太大的差别，bytecode 也是一种中间语言，但在执行过程中二者稍有不同。由于 JVM 采用解释的方法运行 bytecode，所以 Java 程序运行速度较慢，而由 JIT 编译好的本地代码效率更高。但是，因为 JIT 需要在程序第一次执行时进行实时编译，所以程序第一次运行时速度会慢很多。J2EE 同样也支持预编译技术，在用户首次访问某个特定的 JSP 页面时会执行预编译，编译之后，用户再访问这个页面时，速度就会快很多。

通用类型系统（Common Type System，CTS）也可以在 JVM 中找到相对应的内容。CTS 是.NET 语言之间的粘合剂，CTS 对.NET 语言所采用的类型进行了统一的定义来保证语言之间的兼容性。CTS 中定义的类型以及类型的分配方式都与 JVM 非常类似，简单的值类型在栈中分配空间而复杂的引用类型在堆中分配空间。

CLR 和 JVM 的内存管理也非常相近，CLR 和 JVM 都使用了自动垃圾收集来回收不

再被使用的对象。

CLR 出现的时间较 JVM 更晚，其中借鉴了 JVM 中许多闪光的技术点。Microsoft 公司对 CLR 针对 Windows 平台进行了一定的优化，而 JVM 承诺平台无关性，所以比 CLR 的限制更多。总体上来说，二者非常近似。

2.3.2 对多层分布式应用的支持

无论是 J2EE 还是.NET，都非常适合开发企业计算平台，二者都为构建完整的企业计算平台做出了大量的工作，都支持多层应用的开发。例如，在 J2EE 中，开发者可以通过 Java 数据库连接（Java DataBase Connectivity，JDBC）访问数据库，使用 EJB 来编写商业逻辑层，使用 JSP 书写 Web 表现层；在.NET 中，开发者同样可以使用 ADO .NET（ActiveX Data Objects .NET）来访问数据库，使用 C#编写商业逻辑，使用 ASP .NET 来编写 Web 的表现层。SUN 公司和 Microsoft 公司也都提供了使用 J2EE 和.NET 技术构建的电子商务的例子，这些例子都可以在他们的网站中找到相应的源代码。J2EE 和.NET 同时也都支持 Web Service 的开发，为异构系统的应用集成提供了方便。从技术的完备性角度来说，二者不分上下，都提供了全部的必要的技术。下面分表现层、业务层和数据访问层分别讨论 J2EE 和.NET 的实现。

1．表现层

表现层是展现给用户的视图，表现层的好坏直接影响到用户体验。最初，企业应用的表现层基本上都是用本地应用程序作为客户端。随着 B/S 应用的发展，基于浏览器的应用程序越来越多，Web 页面成为了表现层最主要的展现方式。J2EE 和.NET 对这两种形式的表现层都有相应的支持。

对于本地应用程序这种形式的表现层，J2EE 通过抽象窗口工具包（Abstract Windowing Toolkit，AWT）来编写 Java 应用程序，通过 RMI/IIOP 连接业务层构件，或通过简单对象访问协议（Simple Object Access Protocol，SOAP）访问部署成为 Web Service 的业务层构件。与此相对应，.NET 通过 Web Service、.NET Remoting 和 WCF，也可以让本地客户端远程调用业务层构件。在这里，J2EE 与.NET 的最大不同在于，Java 本身是跨平台的设计，因此 Java 应用程序可以运行在 Windows 或其他的平台上，而使用.NET 开发的本地客户端，只能运行在 Windows 平台上。在平台支持的广泛性上，Java 超过了.NET。但从另一个角度来说，Java 无法针对某种特定平台进行优化，只能将各种平台的 GUI 进行抽象，而.NET 则可以获得 Windows 平台最大限度的支持。如果在 Windows 客户端中仅使用最简单的 UI 元素，Java 只能说略逊于.NET。如果对 UI 的要求变高，基于.NET 的 Windows 应用程序无论是在表现能力还是在响应速度上，都会超过 Java 应用程序。在最新的.NET 版本中，微软将 Vista 风格的 UI 引入进来，制定了 WPF。采用 WPF 的 Windows 应用程序可以用“华丽”两个字形容，可以给最终用户惊艳的效果，对提升用户体验有非常大的帮助。除了表现能力外，Java 应用程序的 UI 开发效率与.NET

也有一定的差距，熟悉 Swing（一个用于开发 Java 应用程序 UI 的开发工具包）的开发人员也远不如熟悉 Windows 表单的开发人员数量多，在 Windows 平台上，使用.NET 进行开发相比 Java 开发成本更低。

综上所述，当使用本地应用程序作为客户端时，如果有跨平台的需要，Java 是不二的选择；反之，如果仅适用在 Windows 操作系统中，使用.NET 可以得到更好的效果与更低的开发成本。

对于 Web 表现层，J2EE 从刚刚推出的时候，就将 JSP 和 Servlet 作为标准的一部分，ASP.NET 也是.NET Framework 的重要组成部分。在基于 Web 页面的企业应用中，二者旗鼓相当。随着异步 JavaScript 和 XML（Asynchronous JavaScript And XML，AJAX）技术的引入，Web 页面的表现能力越来越强，用户体验不断提高。在 Web 应用中，J2EE 的优势在于拥有大量的开源项目。基于 Java 的表现层框架层出不穷，Struts、WebWork、Tapestry 等数不胜数。在 Java EE 5 中，新集成进去的 JSF 也是 Web 应用开发的一大利器。在很多时候，Java 开发者更多考虑的不是有没有成熟的表现层框架可用，而是用哪一个更适合目前的需求和团队。相比而言，.NET 领域开源项目很少，有巨大影响力的项目更少，幸好 ASP.NET 本身已经足够完整，可以满足 Web 应用开发的需求。而且 Visual Studio 的集成环境对 Web 开发也提供了强大的支持，图形化设计和代码编写的工作可以在 Visual Studio 中很好的结合到一起，受到了开发者的欢迎。

2．业务层

无论是 Java 还是.NET，业务层都已经非常的成熟。各自都有一整套对象生命周期、事务、内存管理的方法，都有很强的伸缩性。在业务层，J2EE 跨平台的优势得到了充分的体现。对于大型企业而言，PC 服务器往往不能满足应用的需求，经常要用小型机甚至更高档的服务器，如 Power PC、Sparc 等。当使用到非 Windows 操作系统的服务器时，如 AIX、Solaris 等，.NET 失去了竞争的可能，基于 J2EE 的 Websphere、Weblogic 才是流行的解决方案。即使在 PC 服务器上，很多企业也会选用 Linux、FreeBSD 作为操作系统，同样，.NET 是无法部署在这些服务器上的。

除了跨平台的能力外，丰富的 Java 开源项目也给了开发者更多的选择。著名的 Spring 就是一个非常好的轻量级业务层容器，虽然.NET 版的 Spring .NET 也已经推出，但影响力远不如 Java 版的 Spring。Java 的开源项目不仅限于业务层容器，也有丰富的中间件可供选用，如用于作业调度的框架 Quartz、全文搜索引擎 Lucene、工作流引擎 jBPM、缓存管理 JBossCache 等。当然.NET 并不是完全没有对应的解决方案，如作业调用可以使用 Windows 计划任务、全文搜索可以使用 Indexing Service、在.Net 3.0 以后的版本中的 WF 是一个不错的工作流引擎、Enterprise Library 中提供了包括缓存管理在内的一系列构件可以供.NET 程序使用。虽然基于.NET 的开源项目在不断增加，但.NET 开源项目无论是数量还是质量上，目前都很难与 Java 开源相媲美。对于喜欢应用开源项目的组织而言，J2EE 会是更好的选择。

3．数据访问层

数据访问层的主要目标是连接业务构件和数据库，为业务构件提供数据库访问接口和数据持久化的功能。

J2EE 的数据访问层标准是 JDBC，它支持绝大多数的关系型数据库，通过 JDBC，应用程序可以采用与数据库无关的方式来访问、操作各种关系型数据库。

在.NET 中，与 JDBC 相对应的是 ADO .NET。与 JDBC 一样，ADO .NET 也支持绝大多数的关系型数据库，供应用程序使用。

ORM 是一种可以简化对数据库的操作、提高开发效率、提高程序可维护行的技术。在 Java EE 5 中，Java 持久 API（Java Persistence API，JPA）对 ORM 提供了支持，与之相对应，在.NET 中，ADO .NET Entity Framework 同样支持 ORM。除去标准协议的支持外，很多 Java 和.NET 的开源项目也可以实现 ORM，例如，Hibernate、NHibernate 等。

总之，在数据访问层，无论是 J2EE 还是.NET，都有非常多的优秀方案可供选择。

2.3.3 安全性

首先，需要明确一点，没有绝对的安全。任何技术都有瑕疵，都需要在开发过程中谨慎的处理。SUN 公司曾经多次公布过 Java 存在安全漏洞并发布了补丁程序，而.NET 平台发布仅三个月后也在 ASP .NET 构件中发现了安全漏洞。不过 J2EE 和.NET 都在程序的安全问题方面做了大量的工作。

安全性是一个很广泛的话题，其中包含代码安全、安全策略、安全配置、信息保护（加密）、用户的授权和认证等非常多的问题，限于篇幅，本章不详细讨论这些问题。事实上，对于安全策略、安全配置、加密算法的选择、用户授权方式的选择等问题已经不单纯是 J2EE 和.NET 讨论的问题了。这些问题与具体的程序架构、应用需求和用户群都是相关的。开发人员所关心的是，采取相同的架构和策略的系统，采用不同的技术开发会给安全问题带来什么样的影响。因此，本节仅介绍代码的安全性问题。

事实上，在安全方面，.NET 借鉴不少 Java 中的安全机制，以保护代码的安全性。例如，强类型、禁止内存直接访问、对常量访问的控制、缺省的对象初始化过程等。不过，二者在代码检查方面略有差别。

在.NET 中，IL 被编译成为本地代码前，CLR 将对其进行确认和验证。CLR 首先对文件结构和代码的完整性进行检查，然后将对内存和堆栈的安全进行跟踪、对数据的流向和类型的安全进行检查，等等。在程序执行时，除了对方法参数的类型进行检查外，CLR 将不作过多的处理。在.NET 中，除非使用 SkipVerification 选项，否则，默认情况下 CLR 总要进行这些检查来保证代码的安全性。

JVM 负责 Java 类的加载、链接、验证和执行的工作。在 HotSpot 虚拟机中，除了最常用的代码段会被编译并进行优化外，Java 是解释执行的。所以，Java 在运行时进行代码安全性的验证。JVM 使用 bytecode 栈来完成运行时的安全性检查和验证，例如，变量

分配、数据边界、类型转换等。

一般来说，在代码运行前，有些安全性检查是难以完成的，尤其是在输入参数不可预测的情况下，很难彻底的预测代码执行情况。虽然.NET 执行的是静态的代码验证，但是.NET 的验证过程非常全面，同时在程序运行时也会执行一小部分动态验证。所以，无论是.NET 还是 Java，在代码安全性方面都做得非常优秀。

2.3.4　其他特性的比较

本节对 J2EE 和.NET 平台在部署、可移植性、伸缩性和外部支持方面，进行一些简单的比较。

1．部署

部署简单是 Microsoft 公司的一贯作风，Visual Studio 中集成了.NET 应用程序部署和分发的工具。.NET 不支持非 Windows 的操作系统。从另一个角度来说，.NET 应用程序也不需要面对非 Windows 操作系统的环境，这也降低了.NET 部署的风险。

J2EE 应用程序的部署相对复杂，不同的应用服务器会有不同的表现。J2EE 应用程序经常使用在非 Windows 操作系统中，所以部署 J2EE 应用需要考虑不同操作系统的特性，来确定部署的方案。总体上来说，现在 J2EE 应用的部署已经较早期版本的 J2EE 简化了很多，只要处理得当，J2EE 应用的部署还是很容易完成的。

2．可移植性

在可移植性方面，拥有一次编译到处运行的 J2EE 平台无疑是占了上风。采用.NET 平台进行开发，从开发工具、服务器产品到部署环境都需要使用 Microsoft 公司的产品。单一的平台和单一的产品使得整个开发过程没有太多的选择，而 J2EE 则可以根据需要移植到其他的应用服务器或操作系统中。在这里，需要注意的是，将 J2EE 应用程序移植到其他的平台也不是完全不需要做任何修改的。不同厂商和版本的应用服务器和操作系统有着自己的特性，在移植 J2EE 应用前后，要做好移植方案和移植后的测试，这样才能保证 J2EE 应用的顺利移植。

3．伸缩性

伸缩性是企业应用必须考虑的事情，在持续增长的企业中，应用的规模和复杂度也会不断增加，缺乏伸缩性的系统无法面对快速增长的业务量。在伸缩性方面，J2EE 和.NET 都有一系列的解决方案，这些解决方案与应用的具体需求、部署的环境紧密相关。描述这些解决方案不是本书的内容，读者可以通过一些成功案例来了解 J2EE 和.NET 的伸缩性。J2EE 领域的成功案例很多，在 IBM 公司等几个大厂家的支持下，J2EE 广泛应用于金融、电信、保险等大型企业。Myspace.com 使用.NET 平台支撑了亿万用户，也可以证明.NET 平台具有足够的伸缩性。

4．外部支持

在应对企业计算时，J2EE 和.NET 表现得都非常出色，从技术角度来说，J2EE 和.NET

几乎可以完成绝大多数企业应用的需求。

J2EE 得到了很多大厂商的支持，如 IBM 公司和 Oracle 公司等，这些厂商构成了 Java 联盟。IBM 公司的 Websphere，BEA 公司（已被 Oracle 公司收购）的 Weblogic 都是非常优秀的应用服务器，选择这些应用服务器来构建 J2EE 应用，可以得到足够的技术支持。

除了这些大厂商的支持外，Java 开源社区也非常活跃，很多组织为 Java 提供了一个又一个优秀的开源项目。开源的应用服务器、开源的中间件、开源的 J2EE 开发集成环境，等等。只要企业愿意承担使用开源项目的风险，甚至可以全部使用开源项目来构建整个 J2EE 应用。

相比之下，.NET 是 Microsoft 公司独家的产品，使用.NET 意味着必须购买 Microsoft 公司的工具、操作系统、应用服务器和开发工具，没有其他的选择。购买这些产品和工具都是一笔不菲的开支。虽然.NET 开源社区也逐渐活跃起来，但与 Java 相比，还有相当的差距。只能获得一家厂商的支持，是一些企业不愿意使用.NET 的重要原因。

5．小结

由于竞争的关系，J2EE 和.NET 的比较一定会继续下去，而且会持续很长时间。事实上，这两个平台都是非常优秀的。使用 J2EE 和.NET 不但可以进行大型企业级计算系统的开发，同样也适用于中小型应用程序的开发。

J2EE 和.NET 的最大可能就在于 J2EE 是以 SUN 为首的若干公司组成的联盟所大力推广的，而.NET 则是作为 J2EE 的竞争者由 Microsoft 公司开发的。从一定程度上来说，J2EE 平台更开放一些，支持厂商更多一些，同时还有大量的免费产品来搭建 J2EE 平台。但正如 Microsoft 公司把持着.NET 的标准一样，J2EE 的标准也都是由 SUN 制定的。

对于需要进行平台选择的企业和开发者来说，根据自己的实际需求和现状，才能做出最恰当的选择。

本章参考文献

[1] Java EE at a Glance. http://java.sun.com/javaee/
[2] .NET Framework Overiew. http://www.microsoft.com/net/Overview.aspx
[3] 宁建平，梁超. J2EE 参考大全. 北京：电子工业出版社，2003
[4] 李建忠. Microsoft .NET 框架程序设计（修订版）. 北京：清华大学出版社，2003
[5] 谢杨. J2EE 核心技术. http://www.ccw.com.cn/htm/center/tech/02_6_26_8.asp
[6] 柴晓路. J2EE 与.NET 在 Web Services 上的对抗. http://developer.ccidnet.com

第 3 章　中间件技术

随着企业的 IT 环境日益复杂，企业中可能会拥有多种操作系统、不同的数据库、异构的网络环境以及若干应用，那么如何把它们结合成一个有机的协同工作整体，真正实现企业跨平台、分布式应用呢？中间件（Middleware）便是解决之道，它用自己的复杂换取了企业应用的简单。

20 世纪 90 年代初，企业的应用系统开始分层，从最早的单机应用，发展到客户端/服务器结构，然后是三层结构（客户端/应用服务器/数据库服务器），然后是基于浏览器的三层结构，到现在的多层分布式系统结构。在这种多层分布式的系统结构中，各服务器和终端机之间都是通过网络连接起来的，并有大量信息和数据进行传递。对每个应用系统而言，在设计和开发时不仅要关心业务逻辑，还必须要处理分布环境中复杂的通信和异构系统问题，而目前的系统软件（操作系统和支撑软件）还不能满足多样的应用要求。为此，出现了中间件，它是处于系统软件和应用软件之间的一类软件。它使设计者集中设计与应用有关的部分，大大简化了设计和维护工作。通过十多年的大量应用和实践，中间件已有一批成熟的产品，并成为设计分布式系统时不可缺少的部分。现在，中间件已经得到迅猛发展，并逐步走向成熟。

3.1　中间件概述

目前，还没有对中间件形成一个统一的定义，本书使用现在比较认可的两种定义：

（1）中间件是在分布式系统环境中处于操作系统和应用程序之间的软件。

（2）中间件是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源，中间件位于操作系统之上，管理计算资源和网络通信。

读者可以通过图 3-1 来理解中间件在系统中的地位与应用价值。

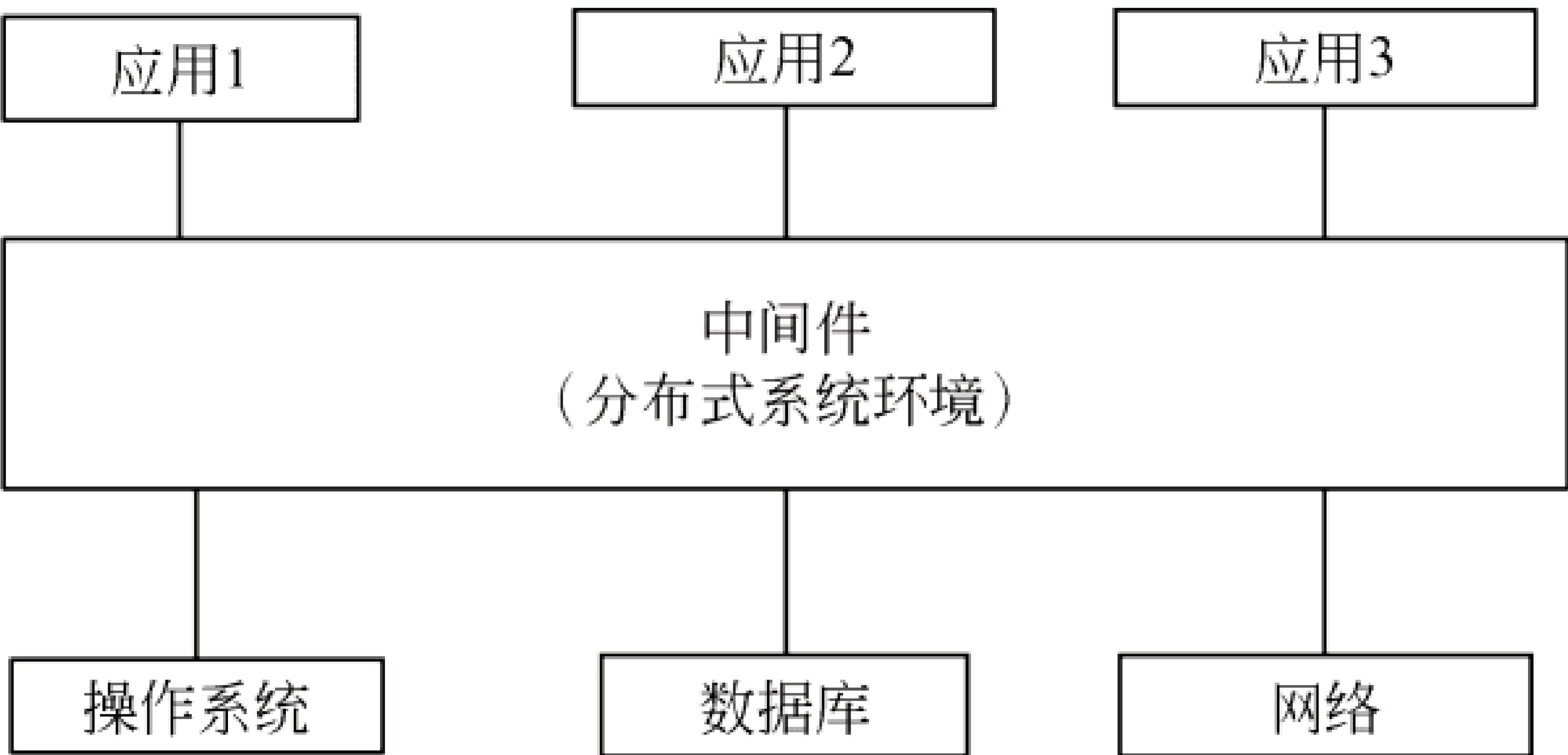


图 3-1　分布式系统中间中示意图

从这些定义中可以看出，中间件具有以下特性：

（1）中间件是一类软件，而非一种软件。

（2）中间件不仅仅实现互连，还要实现应用之间的互操作。

（3）中间件是基于分布式处理的软件，最突出的特点是其网络通信功能。

中间件是处于操作系统和应用程序之间的软件，也有人认为它应该属于操作系统中的一部分。这个定义也限定了只有用于分布式系统中才能称为中间件，同时还可以把它与支撑软件和实用软件区分开来。人们在使用中间件时，往往是一组中间件集成在一起，构成一个平台。随着中间件应用的不断增长，中间件的范围已经覆盖了分布式对象和构件、消息通信、移动应用等软件系统。

具体来说，中间件的基本功能应该包括以下几个：

- 负责客户端和服务器间的联接和通信。
- 提供客户端与应用层的高效率通信机制。
- 提供应用层不同服务之间的互操作机制。
- 提供应用层与数据库之间的联接和控制机制。
- 提供一个多层结构应用开发和运行的平台。
- 提供一个应用开发框架，支持模块化的应用开发。
- 屏蔽硬件、操作系统、网络和数据库。
- 提供交易管理机制，保证交易的一致性。
- 提供应用的负载均衡和高可用性。
- 提供应用的安全机制与管理功能。
- 提供一组通用的服务去执行不同的功能，为的是避免重复的工作和使应用之间可以协作。

3.1.1 中间件的分类

中间件的任务是使应用程序开发变得更容易，通过提供统一的程序抽象，隐藏异构系统和分布式系统下低级别编程的复杂度。中间件分类有很多方式和很多种类型。在这里，由底向上来划分，如图 3-2 所示，可分为底层型中间件、通用型中间件和集成型中间件三个大的层次。

底层型中间件的主流技术主要有 JVM、CLR、自适配通信环境（Adaptive Communication Environment，ACE）、JDBC、开放数据库互连（Open Database Connectivity，ODBC）等，代表产品主要有 SUN JVM、Microsoft CLR；通用型中间件的主流技术主要有 CORBA、J2EE、面向消息的中间件（Message-Oriented Middleware，MOM）、COM、Java 消息服务（Java Messaging Service，JMS）等，代表产品主要有 NA Orbix、BEA WebLogic、IBM MQSeries 等；集成型中间件的主流技术主要有 WorkFlow、企业应用集成（Enterprise Application Integration，EAI）等，代表产品主要有 BEA WebLogic、IBM WebSphere 等。

当然，在这个大的层次划分下，中间件还可以细化为以下一些种类：

（1）通信处理（消息）中间件。在不同平台之间通信，实现分布式系统中可靠、高效、实时的跨平台数据传输的一组软件称为消息中间件。这是中间件中唯一不可缺少的，是需求量最大的中间件产品，目前在大部分操作系统中已包含了其部分功能。

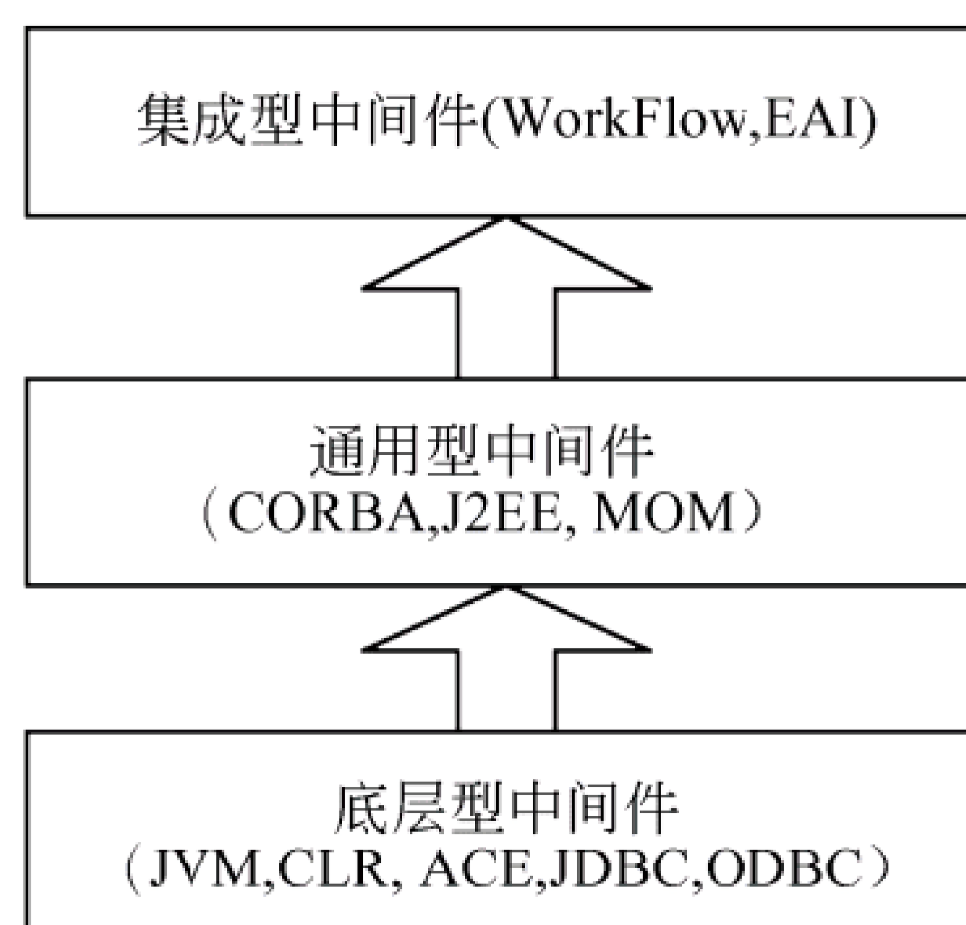


图 3-2　中间件层次图

（2）事务处理（交易）中间件。在分布式事务处理系统中要处理大量事务，常常在系统中要同时进行上万笔事务。在联机事务处理系统（OnLine Transaction Processing，OLTP）中，每笔事务常常要多台服务器上的程序顺序地协调完成，一旦中间发生某种故障时，不但要完成恢复工作，而且要自动切换系统，达到系统永不停机，实现高可靠性运行；同时要使大量事务在多台应用服务器上实时并发运行，并进行负载平衡调度，实现和昂贵的可靠性计算机系统和大型计算机系统同等的功能。为了实现这个目标，就要求事务处理中间件具有监视和调度整个系统的功能。根据 X/OPEN 的参数模型规定，一个事务处理平台应由事务处理中间件、通信处理中间件以及数据存取管理中间件三部分组成。

（3）数据存储管理中间件。在分布式系统中，重要的数据都集中存放在数据库服务器中，它们可以是关系型的、复合文档型、具有各种存放格式的多媒体型，或是经过加密或压缩存放的，该中间件将为在网络上虚拟缓存、格式转换、解压等带来方便。

（4）Web 服务中间件。浏览器图形用户界面已成为公认规范，然而它的会话能力差、不能作数据写入、受 HTTP 协议的限制等，就必需进行修改和扩充，形成了 Web 服务器中间件。

（5）安全中间件。一些军事、政府和商务部门上网的最大障碍是安全保密问题，而且不能使用国外提供的安全措施（如防火墙、加密、认证等），必需用国产的产品。产生不安全因素是由操作系统引起的，这就需要用中间件去解决，以适应灵活多变的要求。

（6）跨平台和架构的中间件。当前开发大型应用软件通常采用基于架构和构件技术，在分布系统中，还需要集成各节点上的不同系统平台上的构件或新老版本的构件，由此产生了架构中间件，功能最强的是 CORBA，可以跨任意平台，但是太庞大；JavaBeans 较灵活简单，很适合于做浏览器，但运行效率差；DCOM 模型主要适合 Windows 平台，已广泛使用。实际上，国内新建系统主要是 UNIX（包括 Linux）和 Windows，因此，针对这两个平台建立相应的中间件要实用得多。

（7）专用平台中间件。为特定应用领域设计参考模式，建立相应架构，配置相应的构件库和中间件，为应用服务器开发和运行特定领域的关键任务（如电子商务、网站等）提供基础。

（8）其他中间件。现在出现了一些中间件，比如数据流中间件、门户中间件，以及为某些专业领域（如银行、电信等）开发的专用中间件。另外，还有一些更高层的中间件，更多用于系统整合，包括 EAI、Workflow、门户（Portal）中间件等是多种中间件的组合。

3.1.2 中间件的优点

中间件作为一大类系统软件，与操作系统、数据库管理系统并称“三套车”，其重要性是不言而喻的，中间件的优点应该说都是有目共睹的。中间件的优越性体现在以下几个方面：

- 缩短应用的开发周期；
- 节约应用的开发成本；
- 减少系统初期的建设成本；
- 降低应用开发的失败率；
- 保护已有的投资；
- 简化应用集成；
- 减少维护费用；
- 提高应用的开发质量；
- 保证技术进步的连续性；
- 增强应用的生命力。

具体地说，中间件屏蔽了底层操作系统的复杂性，使程序开发人员面对一个简单而统一的开发环境，减少了程序设计的复杂性，将注意力集中在自己的业务上，不必再为程序在不同系统软件上的移植而重复工作，从而大大减少了技术上的负担。

中间件带给应用系统的，不只是开发的简便、开发周期的缩短，也减少了系统的维护、运行和管理的工作量，还减少了计算机总体费用的投入。有调查报告显示，由于采用了中间件技术，应用系统的总建设费用可以减少 50%左右。在网络经济大发展、电子商务大发展的今天，从中间件获得利益的不只是 IT 厂商，IT 用户同样是赢家，并且是更有把握的赢家。

其次，中间件作为新层次的基础软件，其重要作用是将不同时期、在不同操作系统上开发的应用软件集成起来，彼此像一个天衣无缝的整体协调工作，这是操作系统、数据库管理系统本身做不了的。中间件的这一作用，使得在技术不断发展之后，用户以往在应用软件上的劳动成果仍然物有所用，节约了大量的人力、财力投入。

3.2 中间件的应用

中间件提供了应用系统基本的运行环境，而中间件服务则提供了更多高级的功能。

如名字服务、事件服务、通告服务、日志等。在这些服务之上，还需要考虑不同行业的需求、不同的应用领域。中间件技术应用层次如图 3-3 所示。

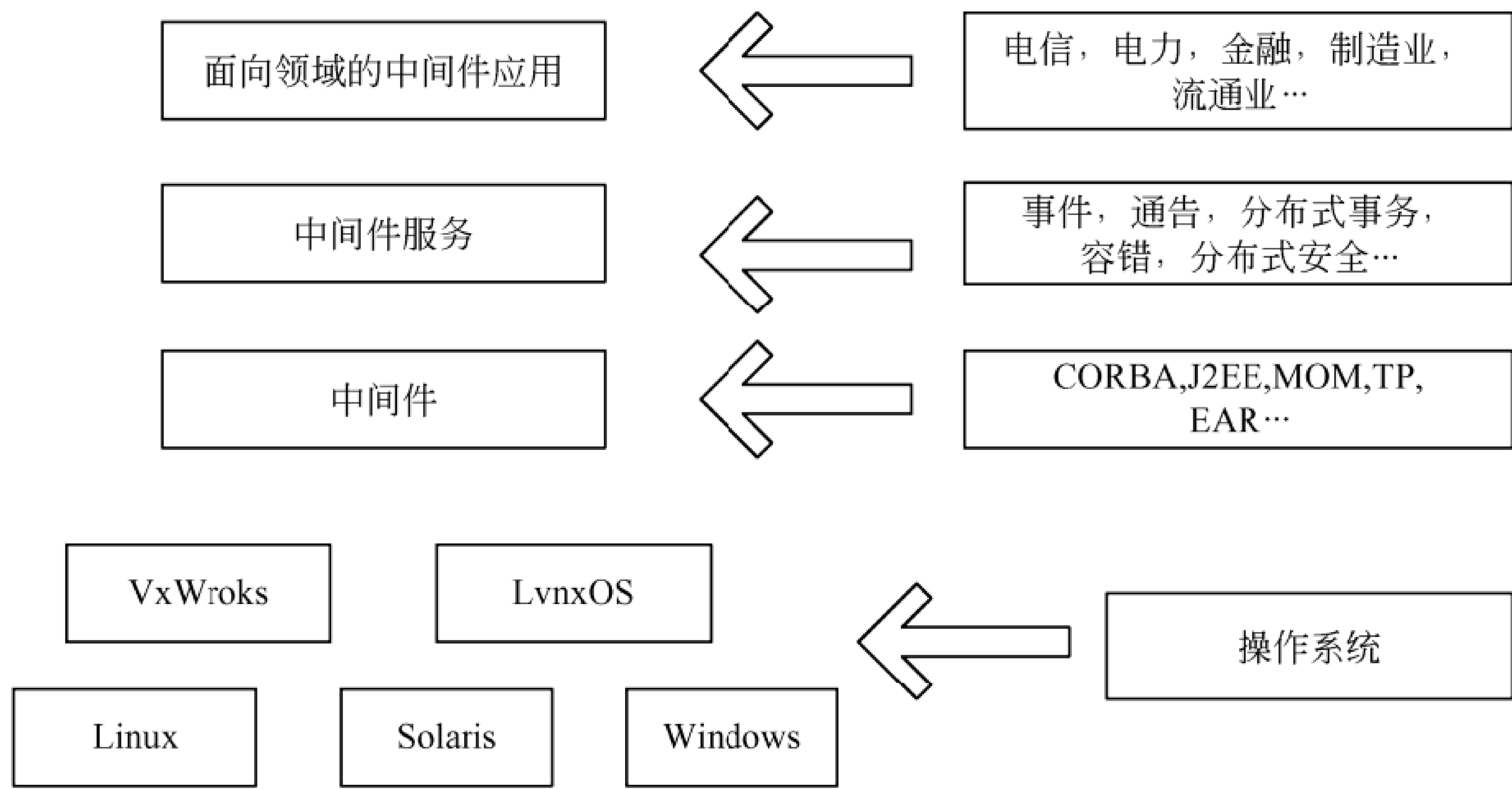


图 3-3　中间件技术应用层次图

3.2.1　中间件技术在集成中的应用

中间件技术在集成中扮演着重要的角色，可以从不同层次采用不同种类，不同技术的中间件产品进行应用集成。正如图 3-4 所示，可以从传输、消息、构件、流程等各个层面分别加以集成。

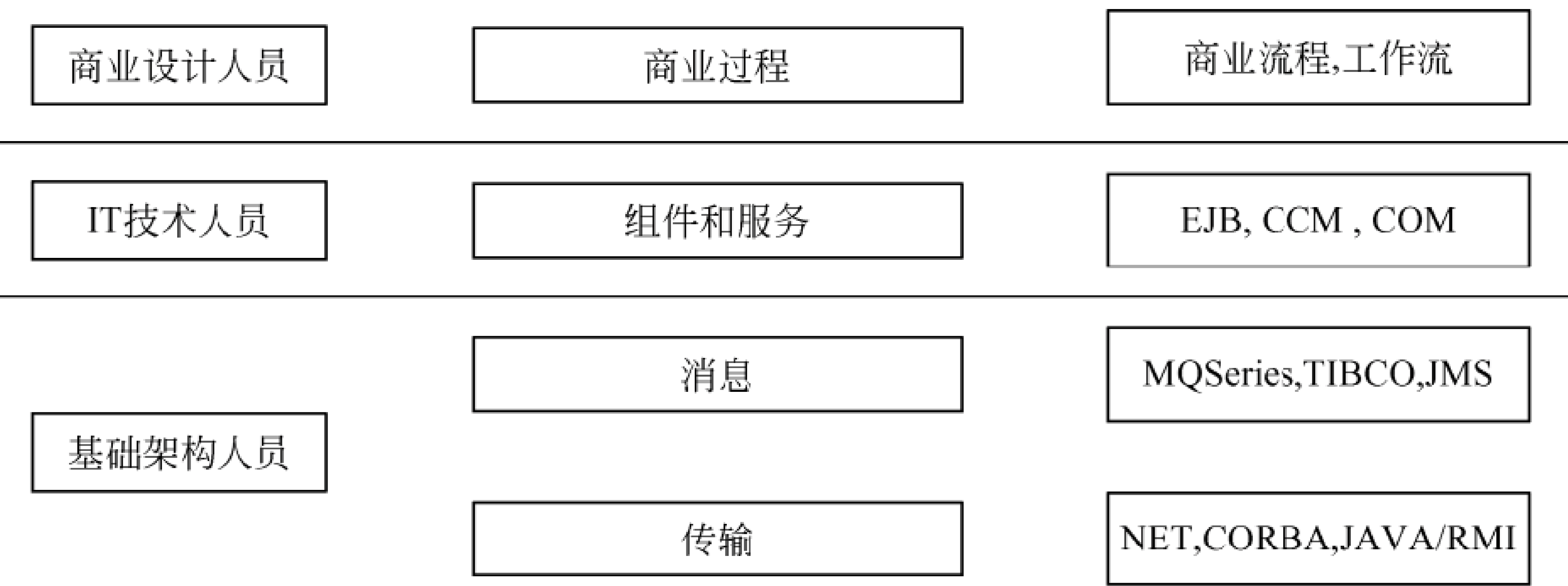


图 3-4　不同层次的集成示意图

从图 3-4 中还可以看出，为了完成不同层次的集成，可以采用不同的技术、产品。例如，为了完成系统底层传输层的集成，可以采用 CORBA 技术；为了完成不同系统的信息传递，可以采用消息中间件产品；为了完成不同硬件和操作系统的集成，可以采用

J2EE 中间件产品。

目前，中间件的竞争焦点主要集中在集成应用平台上。以 TIBCO 为市场领导者的 EAI 市场遭到了来自传统中间件厂商 BEA 等公司的激烈挑战。大多数中间件公司都已经或准备将下一步的工作重点放在了集成市场上。而在集成市场上，Web Services 表现出极强的发展势头。

3.2.2 J2EE 中间件实现

J2EE 是应用服务器采用的主要技术体系，与其他的中间件系统相比较，它具有非常显著的特征，这些特征来自于它独特的体系结构。

企业现在需要通过为他们的客户、合作伙伴、雇员和供应商提供更加便捷的服务来扩大它们的市场，降低它们的成本及缩短它们的响应时间。在许多情况下，可以使用的应用程序必须将现有的企业信息系统与可以为更多的客户提供服务的新业务功能结合起来，这些服务需要：

- 高可用性，以适应当今全球商业环境。
- 安全性，以保护客户的隐私和企业数据的完整性。
- 可靠性和伸缩性，保证事务处理的准确性和及时性。

由于多种原因，这些服务功能需要构筑成有多个层次组成的分布式系统，包括前端的客户端、后端的数据资源端和一个或多个中间层，这个中间层也是开发工作的重点，它实现了新的服务功能和数据与现有业务管理系统的结合。这个中间层将客户层分离出复杂的企业系统，采用先进的 Internet 技术，以减少对客户的管理和培训。

J2EE 降低了开发这些服务功能的成本和复杂性，使服务可以迅速部署，以增强企业回应竞争压力的能力。J2EE 通过以下一些元素定义出一个标准。

- J2EE 平台：一个用于搭建 J2EE 多应用的平台，定义了一组必要的 API 和策略。
- J2EE 兼容性测试套装：一套用于测试 J2EE 平台产品与 J2EE 标准兼容性的测试。
- J2EE 参考实现：一组显示 J2EE 能力的参考实现，也是 J2EE 平台的选择性定义。
- J2EE 设计方针：描述了用于开发中间层、瘦客户应用的标准编程模式。

J2EE 应用服务器由 4 个部分组成，分别是 Applet 容器、客户端容器、Web 容器和 EJB 容器。J2EE 应用服务器由松耦合构件组成，它们协调工作，使多层应用在高性能环境里运行。所有构件都有良好定义的公共接口集和标准实现。这意味着不影响现有应用就可以实现对构件的修改和扩展。为了满足消费者或产品的需要，这种松耦合模型允许开发者修改应用服务器的行为。

应用服务器的核心是基于微内核的。应用服务器的微内核提供了底层的通信、线程、配置、时间、日志等核心功能。在微内核之上，遵循 J2EE 标准实现各种服务。应用在这种微内核的设计模式，使上层标准的服务实现与底层的系统资源管理分离，保持了软件模块间松散耦合的优点。同时，应用服务器还提供了专门的服务接口，允许客户不必

局限在 J2EE 的框架中，直接在内核层次上开发针对于具体案例的系统服务，特别适合于有特殊需求的应用系统。

3.3　中间件与电子商务

电子商务是采用数字化电子方式进行商务数据交换，开展商务业务活动。由于电子商务是在 Internet 等网络上进行的，因此，网络是电子商务最基本的架构。电子商务系统就是商务活动的各方，包括商家、消费者、银行或金融机构、信息公司或证券公司和政府等，中间件技术为全面实现在线交易电子化的过程提供保证。

3.3.1　电子商务中间件架构

从网络环境来看，电子商务所强调的是在网络计算环境下的商业化应用，不仅仅是硬件和软件的结合，也不仅仅是电子交易，而是把买家、卖家、厂商和合作伙伴在因特网（Internet）、企业内部网（Intranet）和企业外部网（Extranet）结合起来的网络应用体系。下面，用图 3-5 对电子商务网络应用体系做个描述。

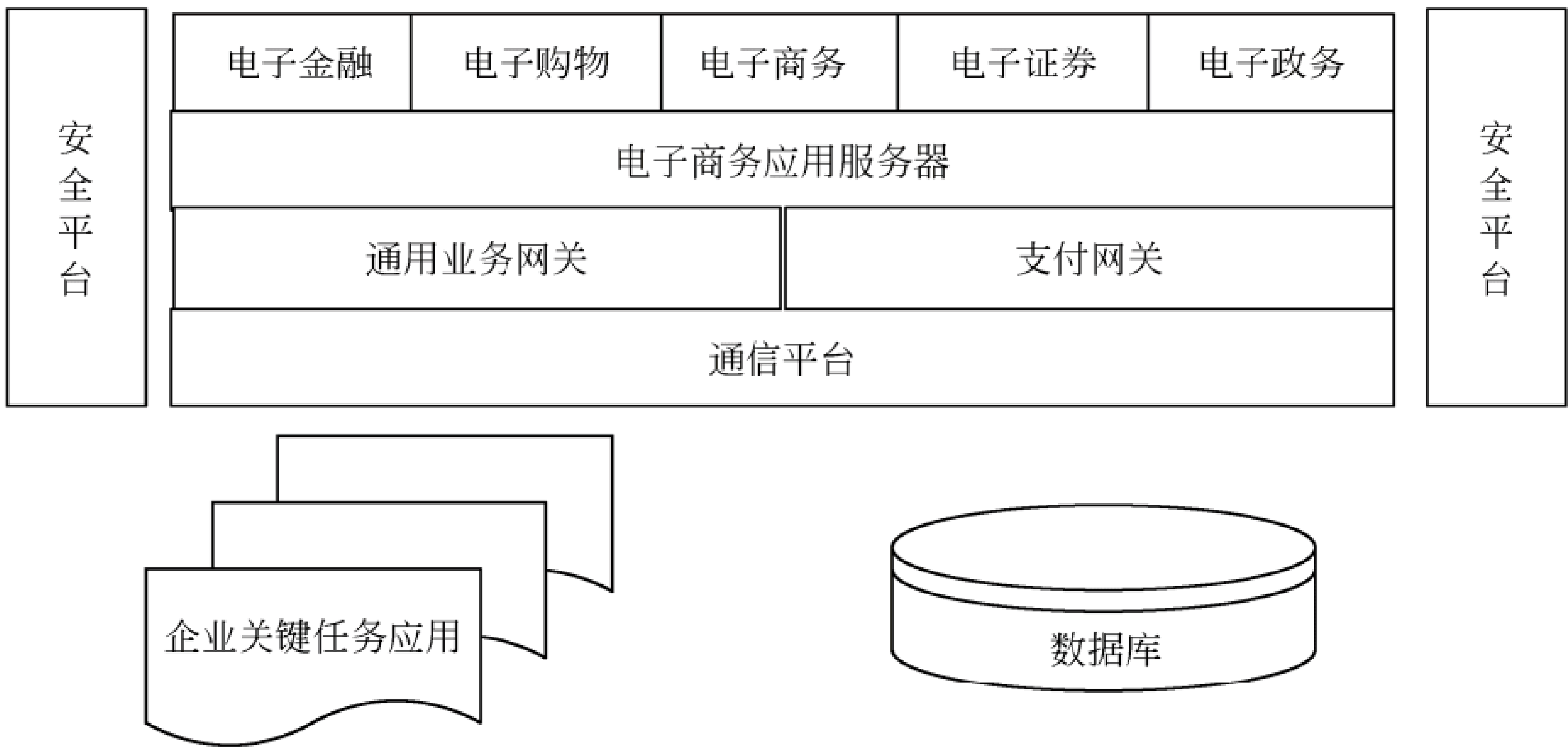


图 3-5　电子商务网络应用体系示意图

简单地说，电子商务网络应用体系包括以下几个方面的内涵。

（1）电子商务应用服务器：是整合事件管理、交易管理、购物管理及供应链管理的框架。

（2）通用业务网关和支付网关：包括动态业务增减机制以及电子支付机制（授权、结算、对账和分账等）。

（3）通信平台：为数据的可靠传输及数据的一致性提供保障。

（4）安全平台：指网络各层次的安全模块，包括安全身份认证、数据加密等。

当然，无论电子商务是怎样一个网络应用体系，其底层仍然离不开作为核心的企业关键任务应用系统和数据库资源。

从应用的角度来看，电子商务网络应用体系的内涵是各种现有应用的不断扩充和新应用形式的不断增加，迫使企业的 IT 部门需要解决越来越多的需求，尤其是对分布式网络应用的需求，诸如跨过不同硬件平台、不同的网络环境、不同的数据库系统之间互操作，新旧系统并存，系统效率过低，传输不可靠、数据需要加密，各种应用模式，开发周期过长，维护不力，等等，这些问题只靠传统的系统软件或 Web 工具软件提供的功能已经不能满足要求，作为电子商务网络应用体系的中间平台也就应运而生了。把电子商务应用服务器、通用业务网关、支付网关、通信平台和安全平台，统一纳入电子商务中间件架构的范畴。

电子商务中间件架构逻辑上位于 Web 服务器之上，负责管理计算资源和网络通信。它是一类软件，而非一种软件；它不仅仅实现互连，还要实现应用之间的互操作与集合。

电子商务中间件架构是一种电子商务应用集成的关键件，不管电子商务应用分布在什么硬件平台上，使用了什么数据库系统，通过了什么复杂的网络，电子商务应用的互连和互操作是电子商务中间件架构首先要解决的问题。在通信方面，电子商务中间件架构要支持各种通信协议和通信服务模式，传输各种数据内容，数据格式翻译、流量控制、数据加密、数据压缩等；在电子商务中间件架构核心部分，要解决名字服务、安全控制、并发控制、可靠性和效率保证等；在电子商务应用开发方面，要能提供基于不同平台的丰富的开发接口，支持流行的开发工具和异构互连接口标准（如 IIOP、DCOM）等；在管理方面，解决电子商务中间件架构本身的配置、监控、调谐，为电子商务应用的易用、易管理提供保证。

其次，针对不同的 Web 应用环境，对电子商务中间件架构有各种不同的要求。对工作流应用，需要根据条件以及条件满足状态，将信息、响应状态从一个应用传递到另一个应用；对联机事务处理，需要保证分布式的数据一致性、不停机作业、大量并发的高效率；对于一个数据采集系统需要保证可靠传输等等。

3.3.2 电子商务应用服务器

电子商务应用服务器的作用是让网络应用的开发、部署、管理变得更加容易，涉及的技术包括 EJB、CORBA、DCOM、IIOP、XML 等。电子商务应用服务器的功能如下：

（1）提供在服务器端的分布式应用的部署，包括对象生命周期管理、线程管理、状态管理、安全管理等。

（2）数据源连接访问管理、交易管理等，可以通过数据访问中间件和交易中间件实现大规模并发网络用户管理、均衡负载、容错等。

（3）与现有系统的无缝连接。

其中，对象生命周期管理、线程管理、状态管理、安全管理、策略管理等，都是由

预置在电子商务应用服务器中的各类服务构件来支持的，在应用的运行效率上有很好的保障，同时大大简化了应用开发的周期与成本。通过基于图形的集中化控制，用户能够随时掌握分布在网络上的众多对象的状态，快速建立对象与对象之间的后援关系，设定对象异常终止时后备对象接替运行的策略。

电子商务应用服务器可以用一种灵活的方式来代表一个商业过程，把商业过程转化到一个包含若干个阶段的框架结构，每一个阶段代表对一个商业对象（如定货单）的分离的操作。在每一个阶段，一个或多个专门的构件对对象进行操作。电子商务应用服务器的另一个重要功能是可以与标准的交易中间件实现集成，这意味着整个平台可以作为一个单一的操作，这在一些必须维护进程的完整性的场合是非常重要的。

3.3.3　通信平台

通信平台作用是建立与维护底层数据通道。在功能上，通信平台提供了一种灵活、可靠的方式，把数据从一个商业伙伴发送给另一个商业伙伴，或把数据从不同的源发地采集到一起或转发。在这里包含了同步/异步传输、通信服务、数据标记、加密、队列和监控，等等。通信平台的主要特性如下：

（1）高效数据通道：前端的大量请求可被汇聚成较少的后端连接并减少数据传送量，保证应用系统即使在大量用户同时请求服务的时候也能够保持快速、稳定的工作状态。

（2）降低网络负担：商务服务器之间建立一条网络通道，多个请求可以复用网络通道。同时，对网络上传递的数据进行压缩，进一步减少网络传递的数据量。

（3）名字服务：提供路由机制，且服务程序可以按优先级进行处理。

（4）支持动态配置，提供系统可伸缩性。

（5）网络故障恢复：自动检测网络连接，如果发现问题，则可以自动重新建立连接。

（6）数据可靠传输：数据传送有可发送时间支持，在设定时间内，如果网络故障得以恢复，则仍然可以正确发送。文件传送支持块重传和断点续传。

希赛教育专家提示：Internet 在国内还存在着多种不可靠因素，如软件不可靠、线路不可靠、系统不可靠等，基础设施方面的问题也可能对性能产生不利影响，如服务器、网卡、总线等跟不上千兆以太网的发展步伐，主存储器及超高速缓存也需要相应的匹配等。因此，必须对通信平台中的可靠队列传输功能提出较高的要求，这就要求使用通信中间件来完成。

3.3.4　安全平台

安全平台是建立在一系列相关国际标准之上的，以公钥算法为核心的一个开放式安全应用开发平台。基于安全平台可以开发、构造各种安全产品或安全应用系统，例如，用于文件加解密的安全工具、安全网关、公证系统、虚拟专用网，以及其他的需要加强安全机制的用户应用系统。

安全平台除了内核的管理模块外，同时向上为应用系统提供开发接口，向下提供统一的密码算法接口及各种 IC 卡、安全芯片等设备的驱动接口。一般来讲，电子商务的安全包括数据的机密性、完整性以及可用性。

数据的机密性指数据传输和存储过程中，采用加密传输，数据不被别人窃取、泄漏、篡改和破坏。如果以加密实现的通信层次来区分，加密可以在通信的三个不同层次来实现，即链路加密、节点加密和端到端加密。

数据的完整性和可用性指数据不会被非授权的用户修改，保持数据一致性。数据的完整性和可用性主要体现在识别机制上，对实体的某些参数进行有效性验证。现在常用的识别技术有报文识别、数字签名和身份识别。其中，身份识别是为电子商务应用系统提供公开密钥基础设施（Public-key Infrastructure，PKI），其核心是密钥及证书的管理。为了确认使用者的真实身份，所有 PKI 的用户必须做事前身份登记，这种登记是以数字化的格式存在，称之为公开密钥证书。针对身份登记所进行的一系列操作与管理，就是所谓的证书管理（Certification Authority，CA），包括用户、过程管理和工具。

其实，网络安全体系很复杂，本节提及的只是有关信息系统在电子商务中所应有的安全性，也就是安全平台所扮演的角色，并不涉及安全策略、物理网络及访问控制（如防火墙、安全访问级别等）。

3.4 构件技术与中间件

中间件作为存在于系统软件与应用之间的特殊层次，抽象了典型的应用模式，从而使应用软件开发者可以更多地将思路放在业务逻辑中，并基于标准的形式进行开发。这样，就使软件架构化成为可能。一些工业标准的推出，进一步使中间件成为可复用构件的运行框架，加速了软件复用的现实化进程。

1. 中间件是构件存在的基础

构件技术在最初时更多是作为一种思想存在，进而才在一些关键的环节上发展出解决问题的技术分支。构件的存在某种程度上极大地依赖了架构技术，或环境、基础设施、计算平台，只有在适当的架构中，软件才有可能被抽象和隔离，最终成为构件。因此，单独讨论构件是抽象而空洞的。架构不是操作系统、数据库或网络协议，也不完全是应用，而是在某种特定意义上的构件运行容器，层次上界于应用和基础设施之间。有关架构的详细知识，将在本书的第 11 章进行介绍。

从本质上来说，中间件是对分布式应用的抽象，因而抛开了与应用相关的业务逻辑的细节，保留了典型的分布交互模式的关键特征。经过抽象，将纷繁复杂的分布式系统经过提炼和必要的隔离后，以统一的层面形式呈现给应用。应用在中间件提供的环境中可以更好地集中于业务逻辑上，并以构件化的形式存在，最终自然而然地在异构环境中实现良好的协同工作。

希赛教育专家提示：中间件与架构实际上是从两种不同的角度看待软件的中间层次。从某种程度上说，中间件就是架构，是构件软件存在的基础，中间件促进了构件化软件。因此可以说，中间件与架构在本质上是一致的。

2．面向需求的构件应用

基于架构的构件化软件开发应当是面向需求的，即设计者集中精力于业务逻辑本身，而不必为分布式应用中的通信、效率、互操作、可靠性、容错性、完整性等大量与业务无直接关系但又非常重要的问题，而耗费大量的精力，理想的架构在这些方面应当为构件软件提供良好的运行环境。事实上，这些正是中间件所要解决的问题，因此，基于中间件开发的应用真正是面向需求的，从本质上符合构件化设计的思想。

3．使业务逻辑容易划分

服务器构件要求有很好的业务自包容性，应用开发者可以按照不同的业务进行功能的划分，体现为不同的接口或交互模式。针对每种业务的设计和开发是可以独立进行的。

架构和中间件有同样的目标：提供业务的分隔和包容性。例如，消息中间件规定了消息是有属性的，其中部分属性则与业务的划分有关，某种服务构件只进行相应类型的消息交互。至于如何保证业务的分类运行与管理，则是中间件的事情。因此可以说，中间件和架构都实现了构件向应用的集成。

4．构件的封装、设计与实现隔离

构件对外发生作用或构件间的交互，都是通过规范定义的接口进行，构件使用者只需要知道构件的接口，而不关心其内部实现，这是设计与实现分开的关键。架构就应当提供构件交互的规则，并基于这些规则实现类似容器的标准环境。

中间件在分布交互模式上都规定了接口（或类似）机制，如接口定义语言（Interface Definition Language，IDL）就是描述接口的语言规范，从早期的分布式计算环境（Distributed Computing Environment，DCE）到现在的 CORBA、DCOM、Java RMI 等都使用 IDL 描述接口，所不同的只是语言规范。客户访问服务（或对象方法）均通过接口进行，至于服务采用怎样的内部实现，基于怎样的语言，甚至怎样的操作系统、数据库，开发者都不用关心。类似地，消息队列也可作为分布交互的手段，消息的语法和语义定义保证了使用与实现的分离，使用消息队列的客户或服务是不依赖于对方的。既然中间件能隔离设计与实现，能在分布的环境中封装实现的细节，那么，基于中间件的构件开发也就是可能的。

5．隔离应用构件与复杂系统资源

架构很重要的一个功能就是将系统资源与应用构件隔离，这是保证构件可重用甚至“即插即用”的基础，与中间件的意图同样是一致的。中间件最大的优势之一就是屏蔽多样的系统资源，保证良好的互操作性。应用构件开发者只需要按照中间件规定的模式进行设计开发，不必考虑下层的系统平台。因此可以说，中间件真正提供了与环境隔离的构件开发模式。

6．符合标准的交互模型

架构不是什么具体软件，而是抽象的模型，但模型中应当定义一些可操作的成分，如标准的协议。标准的中间件则实现了架构的模型，实现了标准的协议。例如，基于CORBA 的对象中间件使用的是 CORBA 规范作为架构模型，具体实现了可互操作的协议，定义了数据表示语法、数据包格式、消息语义等内容。因此，基于中间件的构件是符合标准模型的。

7．软件重用

软件重用（也称为软件复用）是构件化软件生产的根本目标之一，中间件提供了构件封装、构件交互规则、构件与环境的隔离及架构设施等机制，这些都为软件重用提供了方便的解决方案。

另外，通过类似应用桥的机制，中间件可以建立访问过去应用的通道，或在新的中间件体系中建立特殊的运行容器，封装以往的应用，从而最终做到对应用遗产的继承性重用。

8．提供对应用构件的管理

基于中间件的构件软件可以方便地进行管理，因为构件总可以通过方便的标识机制进行划分，还可以使用构件库机制配合一些管理规则。例如，Microsoft 公司的 COM 就是利用 Windows 系统注册表配合几种唯一标识构件的方式，实现构件的登记、注销和定位。CORBA 规范中有接口池、实现池等规范定义，配合应用登记管理的机制，也能对应用构件实施管理。

总之，不难得出结论，基于中间件开发的应用是构件化的，中间件提供了构件的体系结构，大大提高了应用构件生产的效率和质量。

9．构件思想对中间件的作用

中间件本身作为软件产品，正处于方兴未艾之际，因此本身也可以借鉴构件思想，构件化的软件开发对中间件同样适用。

（1）中间件作为分布式计算平台，涉及资源多样，包括各种操作系统、数据库、网络协议甚至语言，其目标是在分布的环境中统一使用这些资源。因此，可以建立针对这些资源的构件库，以动态、灵活的方式进行构件的装配，如针对不同的面向连接的网络协议，可使用统一语义的网络驱动器构件，最灵活的情况是根据配置动态绑定。

（2）中间件的一个重要设计目标是互操作，而互操作的关键是有清晰而与实现无关的接口。因此，在互操作的边界上，必须将构件的思想融入设计中。

（3）中间件的应用范围越来越广，但应用有不同的需要，不同的业务特点，如果仅仅依靠固定的模式去套用，显然不合适。例如，多数管理信息系统（Management Information System，MIS）应用并不需要交易管理；有些分布应用也没有 OLTP 的特点；金融应用中安全就显得十分关键；拓展到 Web 的应用则特别要求精干、安全和适应性强。因此，中间件必须设计成可伸缩的体系，由一些可替换的构件组成，如某些重于可靠，

某些强调实时，某些则需要小巧。产品只有这种定位，才能在变化迅速的市场上总是适应需求，立于不败之地。

（4）中间件不是最终的应用，需要服务于应用开发，但可以面向典型业务的模型，以方便应用的开发，这些模型可以以构件的形式作为产品提供。例如，CORBA 服务和设施就是一些典型应用的抽象体现，使用这些服务的构件，应用可以大大减少开发规模，并获得良好的效果。以架构化技术术语讲，就是领域建模。

（5）成功的商业软件都是非常便于管理的，同样，中间件也有可配置性的需要，管理整个系统是个复杂的行为，但如果转化为若干简单行为的统一，对开发就很简单而明确。事实上，标准的网络管理协议正是蕴涵了这种思想。基于构件化开发的中间件也一样，各个构件自身是独立配置的单元，只需进行集成就可达到系统的管理目标。

因此，构件化的软件设计思想在中间件发展中起到了重要的作用，可以预见，构件化的中间件在今后市场上是有强大生命力的。

构件是一种前沿的软件设计思想，对整个软件行业的发展有着至关重要的推动作用。而中间件作为应用软件系统集成的关键技术，保证了构件化思想的实施，并为构件提供了真正的运行空间。中间件领域工业化标准的制定、统一及实现，使基于构件的应用开发成为可能。反过来，构件对新一代中间件产品中也起到促进作用。

3.5　中间件的发展趋势

中间件作为构筑企业信息系统和电子商务系统的基石和核心技术，向着标准化和构件化方向发展。具体来看，有以下三种发展趋势。

1．规范化

在中间件的发展过程中，做得最好的一件事情就是规范的制定。对于不同类型的中间件，目前都有一些规范可以遵循，如消息类的 JMS，对象类的 CORBA 和 COM/DCOM，交易类的 XA、OTS、JTA/JTS，应用服务器类的 J2EE，数据访问类的 ODBC 和 JDBC，Web Service 有 SOAP、Web Service 描述语言（Web Service Description Language，WSDL）、统一描述、发现和集成（Universal Description Discovery and Integration，UDDI）等。这些规范的建立极大地促进了中间件技术的发展，同时保证了系统的扩展性、开放性和互操作。

2．构件化和松耦合

除了已经得到较为普遍应用的 CORBA、DCOM 等适应 Intranet 的构件技术外，随着企业业务流程整合和电子商务应用的发展，中间件技术朝着面向 Web、松散耦合的方式发展。基于 XML 和 Web 服务的中间件技术，使得不同系统之间、不同应用之间的交互建立在非常灵活的基础上。XML 是一种可扩展的源标识语言，它提供了一种定义新的标识语言标准。XML 技术非常适合于异构系统间的数据交换，因此在国际上已经被普遍

采纳为电子商务的数据标准。而 Web 服务作为基于 Web 技术的构件，在流程中间件的控制和集成下可以灵活、动态地被组织成为跨企业的商务应用。

3．平台化

目前，一些大的中间件厂商在已有的中间件产品基础上，都提出了完整的面向互联网的软件平台战略计划和应用解决方案。SUN 公司是最早提出“网络就是计算机”的公司，它一直致力于向企业提供受到广泛欢迎的网络软件，对互联网的应用和发展发挥了重要作用。IBM 公司提出了面向网络应用的“旧金山计划”，即以 WebSphere、DB2、Tivoli、Domino 四大品牌组成基础架构平台，提供从中间件、服务器到解决方案的一揽子组合服务。Oracle 公司则推出了以 Oracle 9i 为中心的网络软件平台。Microsoft 公司从 2000 年 6 月开始大力宣传“.NET”计划，并作为未来的基本战略，目标是在互联网的基础上，实现所有的计算机群、相关设备和服务协同工作，提供广泛而丰富的解决方案。

本章参考文献

[1] ObjectWeb. 中间件定义. http://middleware.objectweb.org

[2] Wikipedia. 中间件定义. http://en.wikipedia.org/wiki/Middleware

[3] Douglas C Schmidt. C++ Network Programming Mastering Complexity with ACE & Patterns. Addison-Wesley Professional，2001

[4] Chris Britton. IT Architectures and Middleware: Strategies for Building Large, Integrated Systems (2nd Edition). Addison-Wesley Professional，2004

[5] SUN. Sun ONE Application Server 7 Developer's Guide to Web Services. http://docs.sun.com/source/817-2174-10/index.html

[6] CSAI 顾问网. 构件与中间件. http://se.csai.cn/zt/media/index.htm

[7] Paulo Veríssimo, Luís Rodrigues. Distributed Systems for System Architects (Advances in Distributed Computing and Middleware, Volume 1). Springer，2001

第 4 章　Web Service 及其应用

Web Service（Web 服务）是解决应用程序之间相互通信的一项技术。严格地说，Web 服务是描述一系列操作的接口，它使用标准的、规范的 XML 描述接口。这一描述中包括了与服务进行交互所需要的全部细节，包括消息格式、传输协议和服务位置。而在对外的接口中隐藏了服务实现的细节，仅提供一系列可执行的操作，这些操作独立于软、硬件平台和编写服务所用的编程语言。Web Service 即可单独使用，也可与其他 Web Service 一起，实现复杂的商业功能。

抛开这些深奥复杂的定义，用一句话概括目前广泛使用的 Web 应用程序和 Web Service 的区别，那就是：Web 应用程序是面向用户的，而 Web Service 则是面向计算机的。面向计算机的特性赋予了 Web Service 巨大的潜力和广阔的应用空间，Web Service 也因此成为了各大厂商追捧的对象。

4.1　Web Service 概述

来看一个日常办公中的例子。希赛教育通过互联网和公司网站提供了培训视频网上订购的业务，A 公司的业务员小王通过这个平台订购了希赛教育的一些产品，并将这些订购计划输入到 A 公司的信息系统中。于是，小王首先打开浏览器，进入希赛教育网站，在网上填写订单并提交，然后将这份订单按照 A 公司所要求的格式重新填写一遍，再提交到 A 公司的信息系统中。同样的一份订单，虽然表现形式不一样，但所包含的信息是一样的，小王为产生相同的信息而做了两次工作。这种重复的工作不但意味着工作量的增加，也造成更多出错的可能。计算机本身具有很强的数据处理能力，将这种样式转换的工作交给计算机是再合适不过了，但采用传统的 Web 应用程序平台却很难做到这一点。传统的 Web 面向的是用户，如果非要将 Web 页面的订单转换为信息系统中的格式，必须增加新的转换程序，而且对于不同的公司需要不同的转换程序，如果页面样式发生了变化，程序也要作相应的调整；而如果采用 Web Service，则这些问题都可以迎刃而解，通过 Web Service 的一系列技术标准，计算机可以自动地完成转换工作。Web Service 面向计算机和程序的特点可以让程序以更低的代价、更简单的方式集成到一起，降低企业实施电子商务的成本。同时，Web Service 的松散耦合方式也有助于以增量方式开发、部署分布式计算环境。

4.1.1 Web Service 模型

图 4-1 表示了 Web Service 模型的角色及它们之间的操作关系。

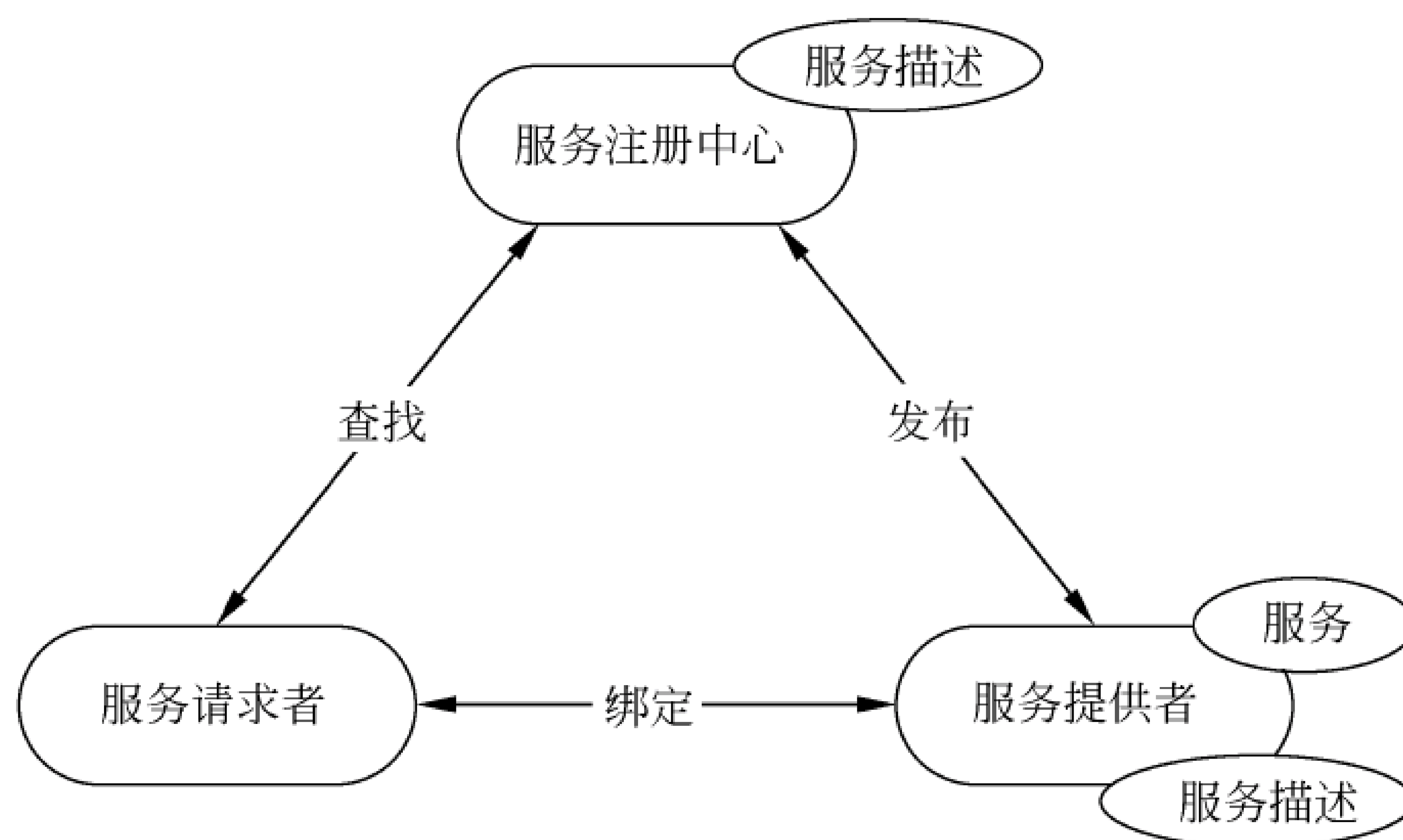


图 4-1 Web Service 模型

1．角色

在 Web Service 模型的解决方案中共有三种工作角色，其中服务提供者（服务器）和服务请求者（客户端）是必须的，服务注册中心是一个可选的角色。它们之间的交互和操作构成了 Web Service 的体系结构。

（1）服务提供者，即 Web Service 的所有者。该角色负责定义并实现 Web Service，使用服务描述语言对 Web Service 进行详细、准确、规范的描述，并将该描述发布到服务注册中心供服务请求者查找并绑定使用。

（2）服务请求者，即 Web Service 的使用者。虽然 Web Service 面向的是程序，但程序的最终使用者仍然是企业或用户。从体系结构的角度看，服务请求者是查找、绑定并调用服务，或与服务进行交互的应用程序。服务请求者角色可以由浏览器来担当，由人或程序（例如，另外一个 Web 服务）来控制。

（3）服务注册中心。服务注册中心是连接服务提供者和服务请求者的纽带，服务提供者在此发布他们的服务描述，而服务请求者在服务注册中心查找他们需要的 Web 服务。不过，在某些情况下，服务注册中心是整个模型中的可选角色。例如，使用静态绑定的 Web Service，服务提供者可以把描述直接发送给服务请求者。在没有服务注册中心的 Web Service 中服务请求者可以从其他来源得到服务描述，例如，文件、文件传输协议（File Transfer Protocol，FTP）站点、Web 站点、广告和服务发现（Advertisement and Discovery of Services，ADS）或发现 Web 服务（Discovery of Web Services，DISCO）等。

2．操作

Web Service 模型中的操作主要有以下三种：发布服务描述、查找服务描述、根据服务描述绑定或调用服务。这些操作可以单次或反复出现。

（1）发布。为了使用户能够访问 Web Service，服务提供者需要发布服务描述使得服务请求者可以查找它。

（2）查找。在查找操作中，服务请求者直接检索服务描述或在服务注册中心查询所要求的服务类型。对于服务请求者，可能会在生命周期的两个不同阶段中牵涉到查找操作，它们分别是：在设计阶段，为了程序开发而查找服务的接口描述；在运行阶段，为了调用而查找服务的位置描述。

（3）绑定。在绑定操作中，服务请求者使用服务描述中的绑定细节来定位、联系并调用服务，从而在运行时与服务进行交互。绑定可以分为动态绑定和静态绑定。在动态绑定中，服务请求者通过服务注册中心查找服务描述，并动态地与 Web Service 交互；在静态绑定中，服务请求者实际上已经与服务提供者达成默契，通过本地文件或其他方式直接与 Web Service 进行绑定。

3．流程

目前，Internet 上已经有了不少 Web Service 可供使用。那么，该如何使用这些 Web Service 呢？图 4-2 就是万维网联盟（World Wide Web Consortium，W3C）所制定的 Web Service 使用流程标准。

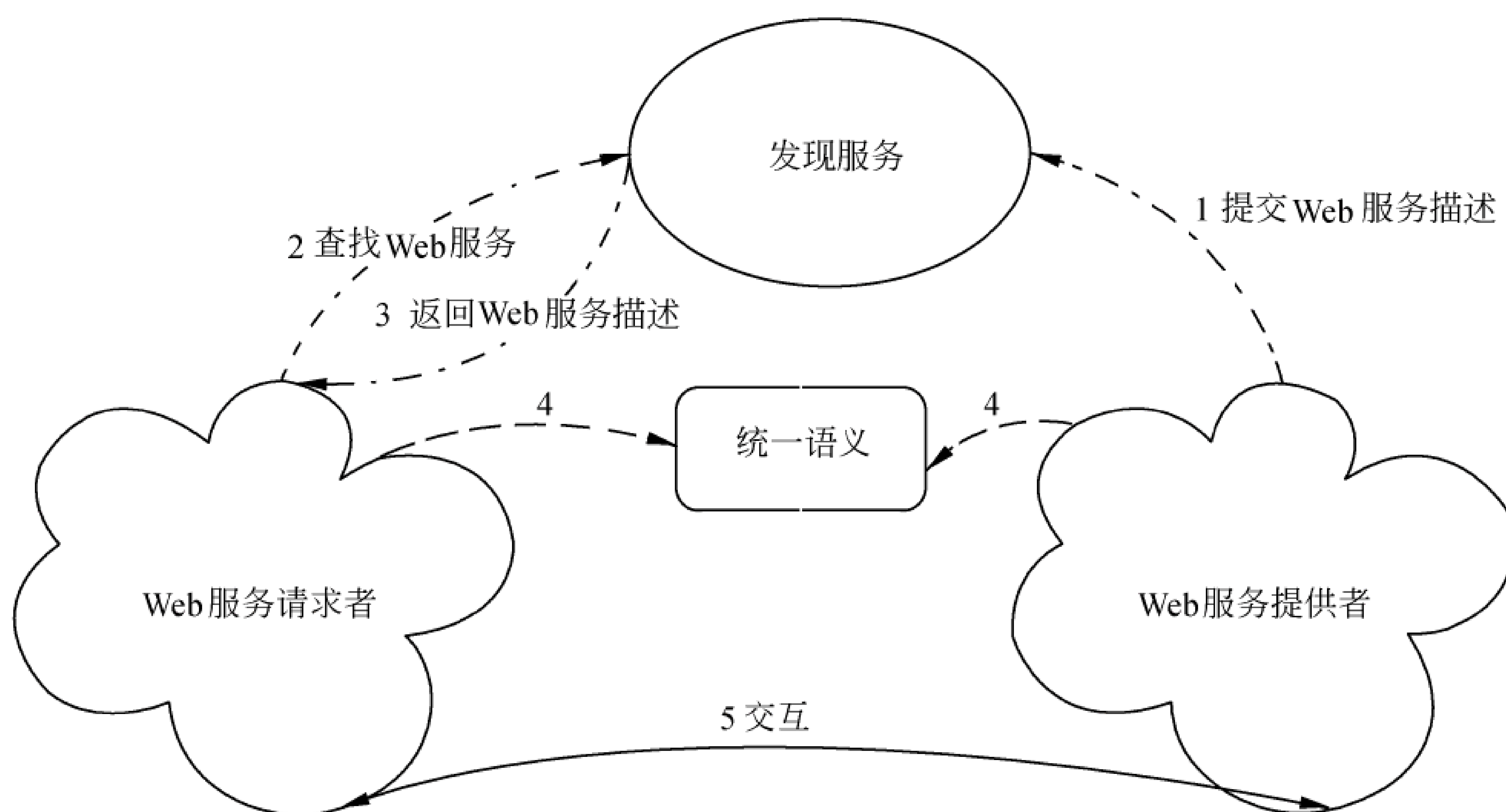


图 4-2　Web Service 使用流程

在图 4-2 的每一道线中都标有一个数字，数字的大小代表了消息发生的先后顺序。在使用 Web Service 时，首先需要服务提供者将 Web Service 的描述信息提交到服务注册

中心（也就是 4-2 图中所谓的发现服务中）。当服务请求者需要使用 Web Service 时，它将首先通过发现服务查找需要的 Web Service，这就是图中的第二步。当找到合适的 Web Service 后，发现服务将返回请求者所需要的 Web 服务描述。在此之后，服务请求者并不是马上就与服务提供者进行 Web Service 的调用，它首先需要与服务提供者统一各自的语义，以保证可以相互理解对方的请求和响应。当然，服务请求者可以按照服务提供者规定的语义信息进行服务调用，不过更合理的做法是双方遵循一个共同的行业标准，这个标准可以由一些相关的行业协会制定。当一切准备工作都完成后，服务者就可以直接与 Web 服务提供者进行交互，调用 Web Service。

4.1.2　Web Service 协议堆栈

Web Service 的使用涉及到很多协议，图 4-3 是 W3C 在 2004 年 2 月 11 日提出的 Web Service 协议栈。

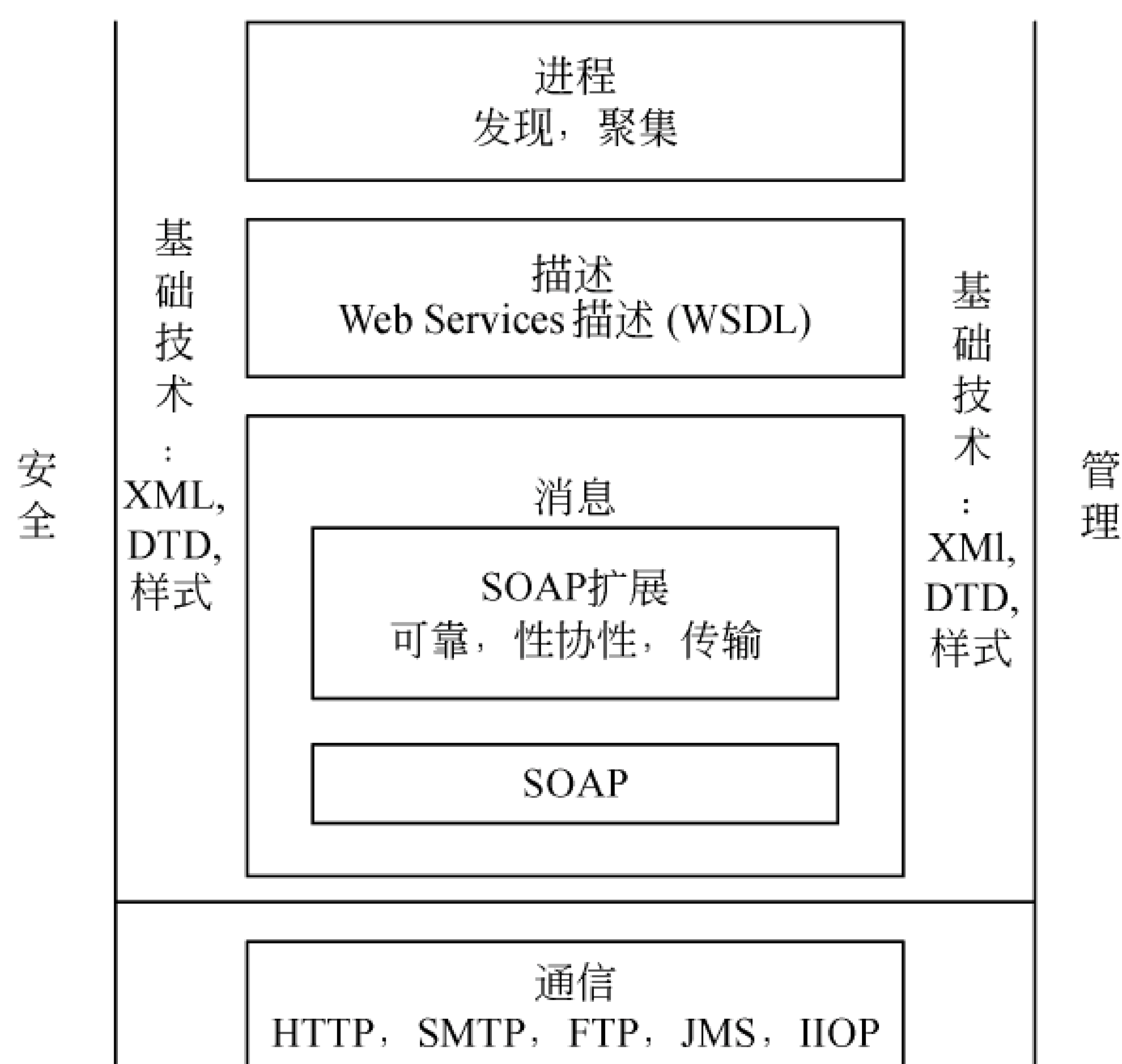


图 4-3　Web Service 协议栈

在协议堆栈的下层为网络通信部分，Web Service 继承了 Web 的访问方式，使用 HTTP（S）作为网络传输的基础，除此之外 Web Service 还采用了其他的传输协议如 SMTP、FTP、JMS、IIOP 等。在消息处理方面，Web Service 使用了 SOAP 作为消息的传送标准。在此之上是 Web Service 描述语言 WSDL，用以描述 Web Service 的访问方法。位于最顶层的是与 Web Service 和应用程序以及 Web Service 之间相互集成相关的协议，其中包含发现、集成等若干方面。在这一层，将介绍 UDDI 协议，UDDI 也是 Web Service 领域中赫赫有名的动态发现协议。除了底层的传输协议外，整个 Web Service 协议栈是

以 XML 为基础的，XML 语义的精确性和灵活性赋予了 Web Service 强大的功能。除这些基本协议外，还有一些需要讨论的问题，那就是安全和管理，这两大问题不是 Web Service 可以独立解决的。如在安全方面就需要与 PKI、轻量目录访问协议（Lightweight Directory Access Protocol，LDAP）等相结合。

1．SOAP

SOAP 是一种基于 XML 的协议，通过 SOAP，应用程序可以在网络中进行数据交换和远程调用。图 4-4 显示了 SOAP 在网络应用程序之间交换数据的方式。

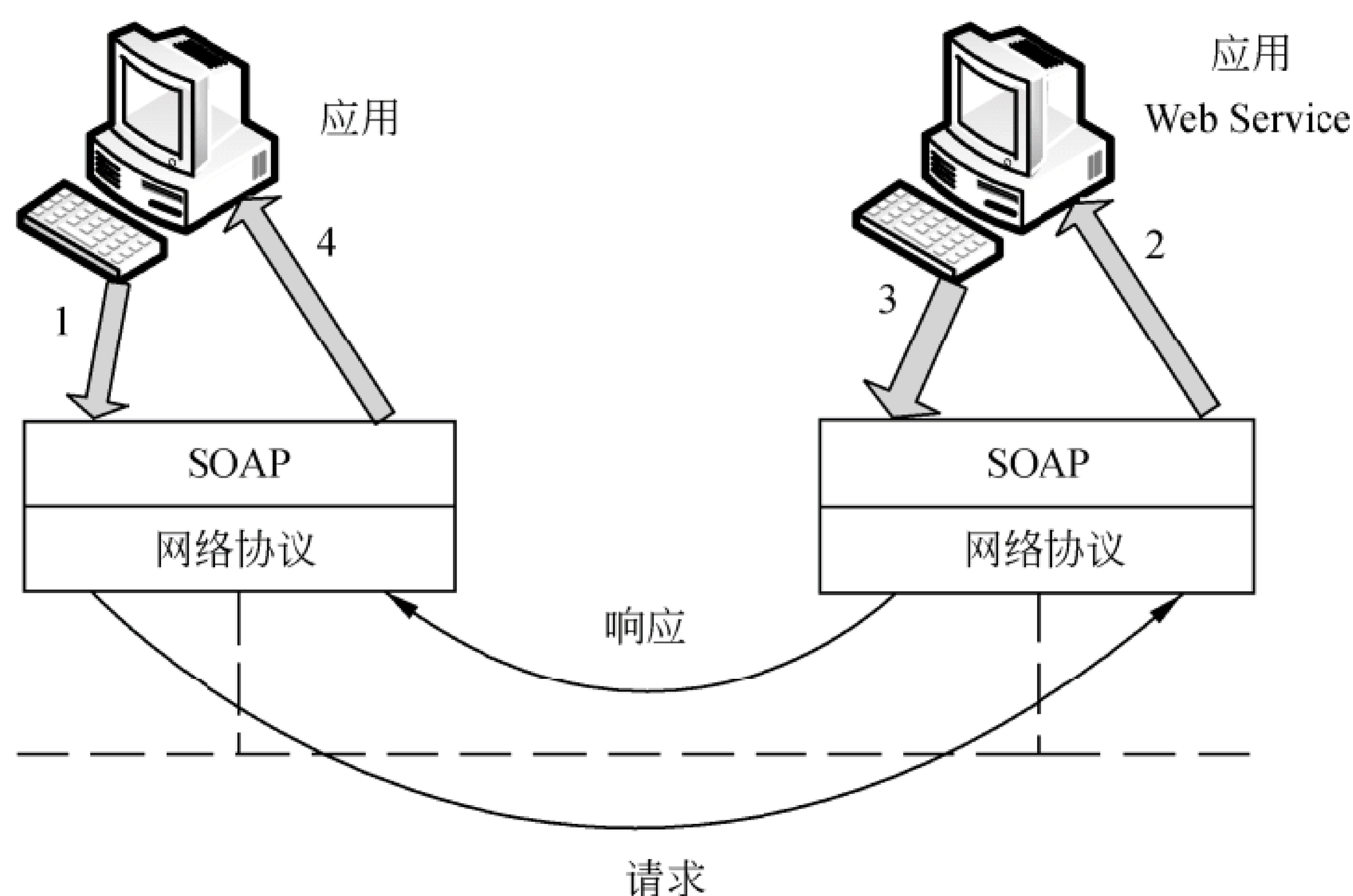


图 4-4　SOAP 工作模式

看到这里，有的读者会问：SOAP 实质上是一个基于 XML 的 RPC 标准，它与同为 RPC 标准的 CORBA、COM/DCOM 有什么区别呢？首先，看一看什么是 CORBA 和 COM/DCOM。

CORBA 是一种标准的面向对象应用程序的体系规范。由对象请求代理 ORB（Object Request Broker）、对象服务、公共设施、域接口和应用接口这几个部分组成。其核心是对象请求代理 ORB，ORB 提供了一种机制，使对象可以透明的发出请求和接收响应。分布的互操作对象可以利用 ORB 构造互操作应用。ORB 可看作是在对象之间建立客户/服务关系的中间件。基于 ORB，客户可以透明的调用服务对象提供的方法。该服务对象可以与客户运行在同一台机器上，也可以运行在其他机器上通过网络与客户进行交互。ORB 截取客户发送的请求，并负责在该软件总线上找到实现该请求的服务对象，然后完成参数、方法调用，并返回最终结果。

COM/DCOM 是 Microsoft 公司提出的分布式构件对象模型标准，支持在局域网、广域网甚至 Internet 上不同计算机对象之间的通信。DCOM 基于 COM 的应用程序、构件、工具等，来处理网络协议低层次的细节问题，因而本身不必关心太多的网络协议细节，

使用户能够集中精力解决自身所要求的问题。DCOM 位于应用程序的构件之间，将构件以不可见的方式组合在一起，形成具有完整功能的应用程序。

明确了 CORBA 和 COM/DCOM 的概念以后，立即可以明确的是：SOAP 与 CORBA、COM/DCOM 不是同一层面上的概念。SOAP 是一个基于 XML 的分布式对象通信协议，CORBA 是分布式应用的服务标准，COM/DCOM 则是一个构件模型。无论是 CORBA 还是 DCOM 都可以使用 SOAP 作为分布式对象通信的标准。除了概念上的区别外，SOAP 和 CORBA、COM/DCOM 还有更多的差异，例如：

（1）采用 CORBA 或 COM/DCOM 构造的应用程序不能混用，二者不能协作。但 SOAP 却可以在二者之间建立联系，实现 CORBA 和 DCOM 的整合。

（2）SOAP 使用 XML 进行编码，是一个开放式的协议。SOAP 本身并没有定义信息的语义、服务质量、事务处理等问题，但 CORBA 和 DCOM 对这些问题都有相应的约定。

（3）SOAP 仅仅是一个对象通信协议，类似于 CORBA 中的 IIOP，是一个层次较低的协议。相比起来，CORBA 和 COM/DCOM 协议则复杂得多。

（4）SOAP 是与应用平台完全无关的。虽然 CORBA 也可以在各种平台上运行，但 CORBA 只能与采用 CORBA 标准的应用程序通信；而 COM/DCOM 则只能在微软的平台中应用。

由于 SOAP 采用 XML 和 HTTP 封装通信消息，所以 SOAP 需要增加 XML 解析和 HTTP 传输的额外开销。但是 SOAP 同时也继承了 XML 和 HTTP 的优点，严格的语法格式使 XML 在 Internet 中得到了广泛应用。

2．WSDL

大家都知道，在通常的开发过程中，对象接口一定具备相应的 SDK 描述文档，Web 服务也是一种对象，是一种被部署在 Web 上的对象。很自然，也完全需要有对 Web 服务这个对象进行描述的 SDK 文档。

当然，这两者不完全相同。第一，目前在 Web 上的应用已经完全接受了 XML 这个基本的标准，WSDL 是基于 XML 的标准；第二，Web 服务的目标是即时装配、松散耦合以及自动集成，这意味着 SDK 描述文档应当具备被机器识别的能力。也就是说，对于使用标准化的消息格式和通信协议的 Web 服务，它需要以某种结构化的方式对 Web 服务的调用和通信加以描述，实现这一点显然非常重要，这是 Web 服务即时装配的基本保证。WSDL 包含了一套基于 XML 的语法，将 Web 服务描述为能够进行消息交换的服务访问点的集合，从而满足了这种需求。WSDL 定义了可被机器识别的 SDK 文档，同时 WSDL 也可用于描述自动执行应用程序在通信中所涉及的细节问题。

希赛教育专家提示：WSDL 的目标是描述如何使用程序来调用 Web 服务，所以，可以把 WSDL 理解为 Web Service 的 SDK 标准，或是 Web Service 的接口定义。对于服务提供者来说，他们既需要描述他们提供的 Web Service 是做什么的，还要描述如何使用他们提供的 Web Service。对于这些问题的具体描述将在 4.2.1 节中详细讲述。

3．UDDI

UDDI 提供了一种 Web Service 的发布、查找和定位的方法。可以将 UDDI 理解为一种目录服务，Web Service 提供者使用 UDDI 将服务发布到服务注册中心，而 Web Service 使用者通过 UDDI 查找并定位服务。UDDI 除了目录服务之外，还定义了一个用 XML 表示的服务描述标准。在讨论了 SOAP 之后，读者可以知道，通过 SOAP 和 XML 可以很方便地将不同企业的不同应用集成到一起，但仅仅有 SOAP 和 XML 仍然是不够的。SOAP 和 XML 不能够提供任何计算机平台都能支持的端到端的解决方案。UDDI 在 SOAP 和 XML 之上建立了新的层次，通过 UDDI 不同的企业可以以统一的标准描述自己的 Web 服务，或查询其他的服务。

除了使用 UDDI 发布和发现 Web 服务外，还有其他几种服务的发布方式，其中最简单的方式是直接发布。直接发布就是服务提供者直接将服务通过使用电子邮件附件、FTP 站点甚至光盘发布给服务请求者。直接发布一般是企业双方在使用电子商务的各项条款达成一致后进行，或在请求访问服务的服务请求者支付了费用之后进行。在这种情况下，服务请求者可以保留服务描述的一份本地副本。很明显，直接发布是一种静态的服务发布方式，灵活性很差。

动态发布方法有 DISCO 和 ADS。DISCO 和 ADS 两者都定义了一个从给定统一资源定位符（Uniform Resource Locator，URL）获取 Web 服务描述的机制。然而，这种简单的获取服务描述的方法也不能够完全满足动态的电子商务模式的要求，UDDI 的功能要强大得多。

UDDI 定义了一种 Web Service 的发布方式。首先，UDDI 注册中心可以为程序或程序员提供 Web Service 的位置和技术信息。服务提供者可以向专用的 UDDI 节点发布服务的描述信息，而服务的使用者可以动态地查询并连接到特定的 Web Service。可以将这几种服务发布技术放到坐标系中，如图 4-5 所示。纵坐标度量服务发布的能力，横坐标度量发布的灵活度。从图 4-5 中可以看出，UDDI 的发布方式是功能最强大、灵活度最高的。

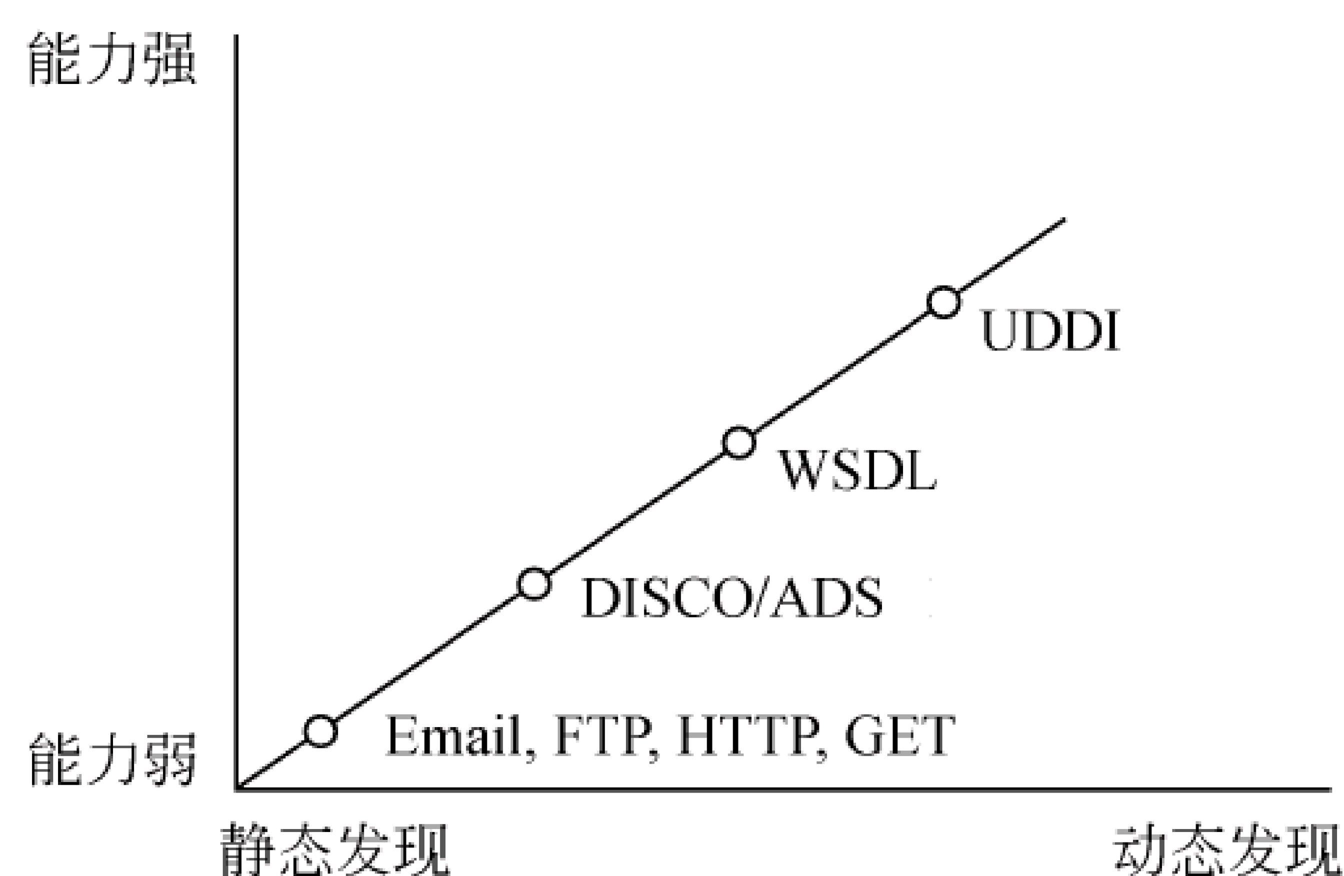


图 4-5　服务发布方式比较

4．XML

XML 在 Web Service 中有着非常重要的应用，可以毫不夸张地说，没有 XML 技术标准，就没有 Web Service 的出现。

Web Service 是一种部署在 Web 上的对象或构件，人们期望通过 Web Service 实现松散耦合的分布式构件互联，以适应 Internet 的计算环境。当讨论这种分布式互联策略时，遇到的最大的问题就是如何将形态各异的数据结构、程序接口、操作系统、硬件平台有

效地结合到一起。XML 恰好就是解决这一问题的利器，XML 具有严密的数据格式和灵活的表现方式，便于数据传输、转换和表现，因此 SOAP、UDDI 和 WSDL 都是在 XML 基础之上定义的。

第 10 章将详细介绍 XML 方面的知识，在此不再赘述。

4.2 WSDL

严格地说，Web Service 是一个协议栈，其中包含了很多相关的协议。这些协议包含对 Web 服务的调用、描述、发布、寻找、管理、安全等。其中最重要的也是最成熟的三个协议是 SOAP、WSDL 和 UDDI。掌握了这三个协议，就可以构造完整的 Web 服务了。本节将详细介绍 WSDL 协议，4.3 节将详细介绍 UDDI 协议，4.4 节将详细介绍 SOAP 协议。

为了方便，在讲述协议之前，先给出一个具体的 Web Service 的例子，后面对于 WSDL、UDDI 和构造 Web Service 的方法会用到这个例子。许多经典的语言教程都是以“Hello World”开始的，将第一个 Web Service 设计为“Hello your name”。这个 Web Service 需要服务调用者传入 string 类型的参数，并返回一个形式为“Hello <string>”的字符串。本节将给出实现该 Web Service 的方法，4.3 节将编写一个调用该服务的小程序，4.5 节将给出这个 Web Service 的实现代码。

4.2.1 WSDL 概述

WSDL 是对 Web 服务进行描述的语言，它有一套基于 XML 的语法定义。WSDL 描述的重点是服务，它包含服务实现定义（Service Implementation Definition）和服务接口定义（Service Interface Definition），如图 4-6 所示。

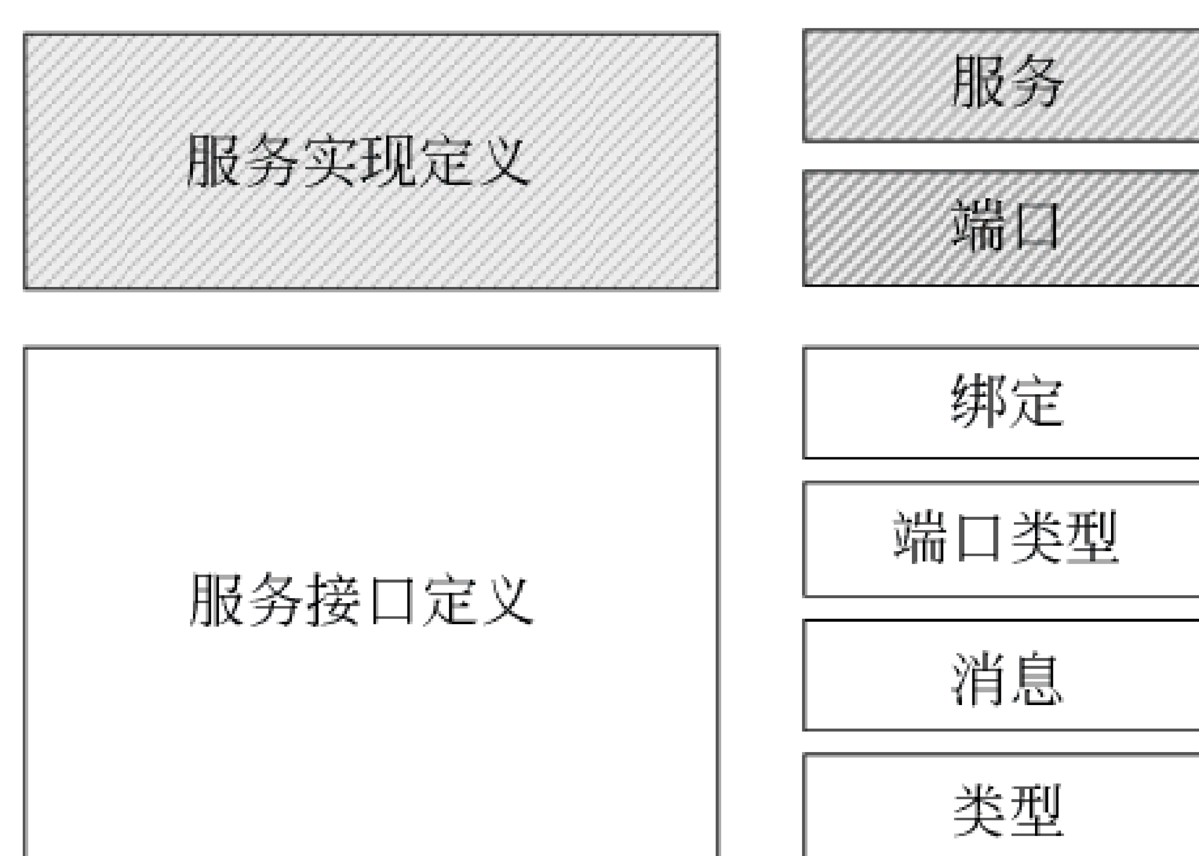


图 4-6 基本服务描述

采用抽象接口定义对于提高系统的扩展性很有帮助。服务接口定义就是一种抽象的、可重用的定义，行业标准组织可以使用这种抽象的定义来规定一些标准的服务类型，服务实现者可以根据这些标准定义来实现具体的服务。这就好比在 Java 中定义的一个抽象接口可以有多个实现一样。

服务实现定义描述了给定服务提供者如何实现特定的服务接口。服务实现定义中包含服务和端口描述。服务描述了一个特定的 Web 服务所提供的所有访问入口的部署细节，一个服务往往会包含多个服务访问入口，而每个访问入口都会使用一个端口元素来描述；端口描述的是一个服务访问入口的部署细节，包括通过哪个 URL 来访问，应当

使用怎样的消息调用模式来访问等。

服务接口定义和服务实现定义结合在一起，组成了完整的 WSDL 定义。

4.2.2　使用 WSDL 文档

WSDL 文档是按照 WSDL 语法规范描述某个特定 Web 服务的文档。WSDL 文档是一个简单的 XML 文档，有一些工具可以自动生成 Web Service 的 WSDL 文档，开发者也可以根据自己的需要书写 WSDL 文档。

虽然说可以把 WSDL 文档看作是 Web Service 的 SDK，但开发者并不直接使用 WSDL 文档，WSDL 文档是由程序使用的，这也是 Web Service 的重要优点。程序级的利用可以让开发者书写出更灵活的代码，更便于不同程序之间的集成。

具体来说，有两个不同的时期来使用 WSDL 文档，一个是在编写调用 Web Service 的客户端程序的设计时期，一个是在程序运行时期。

在设计时期，开发者根据 WSDL 文档生成调用 Web 服务的客户端代码，可以将这样的代码称为 Web Service 代理类，实际的 Web 服务调用都发生在代理类与 Web Service 之间。在编写客户应用程序的过程中，开发者就像使用本地类一样直接使用 Web Service 代理类，开发客户端程序。图 4-7 表明了这种使用方式的流程。

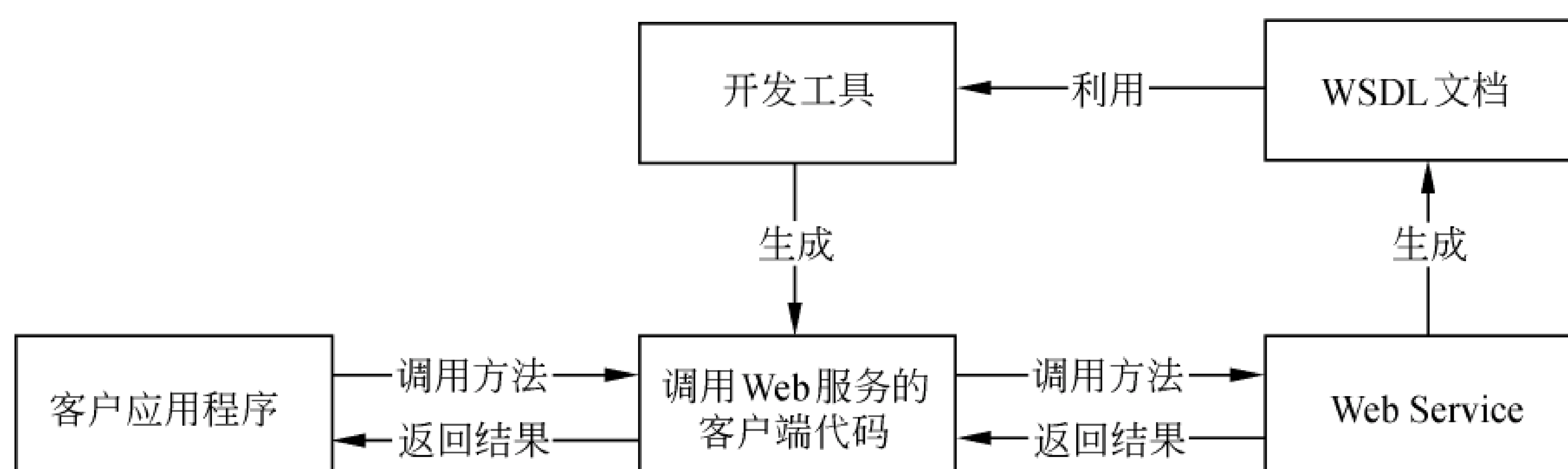


图 4-7　设计期 WSDL 文档的使用方法

在运行期，客户应用程序通过它所处的运行环境发出对 Web Service 的调用请求，运行环境根据 WSDL 文档生成正确的 Web Service 调用请求，并接受 Web Service 返回的结果，然后把结果传递给客户应用程序。图 4-8 表明了这种使用方法的流程。

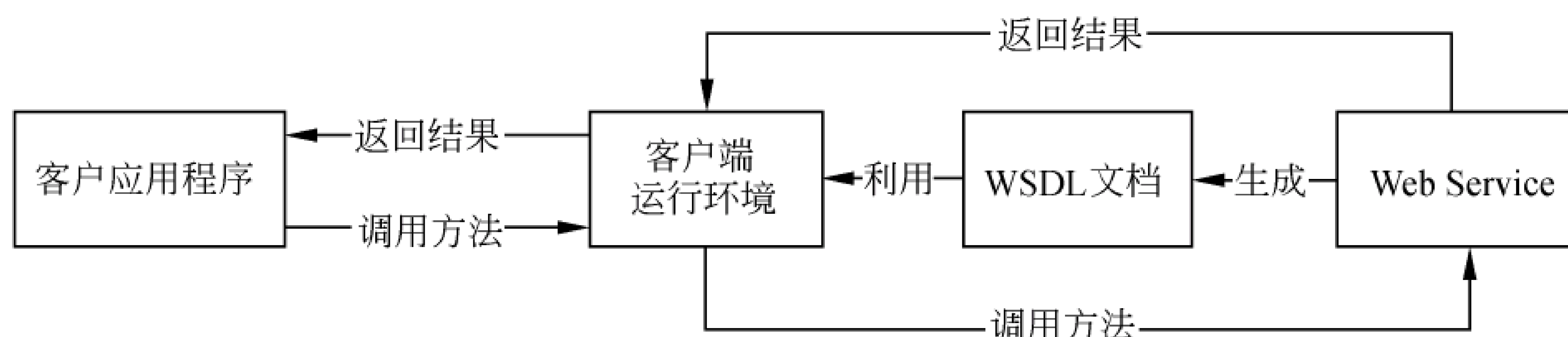


图 4-8　运行期 WSDL 文档的使用方法

4.2.3 WSDL 文档结构

WSDL 文档是一个按照严格规范完成的 XML 文档，WSDL 规范也就是这个 XML 文档的规范。一个标准的 WSDL 文档形式如下：

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri">
<import namespace="uri" location="url"/>*

<wsdl:documentation .../>?

<wsdl:types> ?
    <wsdl:documentation ... />?
    <xsd:schema .../>*
    <--extensibility element -->*
</wsdl:types>

<wsdl:message name ="nmtoken">*
    <wsdl:documentation ... />?
    <part name ="nmtoken" element ="qname"? type ="qname"?/>*
</wsdl:message>

<wsdl:portType name="nmtoken">*
    <wsdl:documentation ... />?
    <wsdl:operation name ="nmtoken">*
        <wsdl:documentation ... />?
        <wsdl:input name="nmtoken"? message="qname">?
            <wsdl:documentation ... />?
        </wsdl:input>
        <wsdl:output name="nmtoken"? message="qname">?
            <wsdl:documentation ... />?
        </wsdl:output>
        <wsdl:fault name="nmtoken"? message="qname">?
            <wsdl:documentation ... />?
        </wsdl:fault>
    </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="nmtoken" type="qname">*
    <wsdl:documentation ... />?
    <-- extensibility element -->*
```

```
        <wsdl:operation name ="nmtoken">*
            <wsdl:documentation ... />?
            <-- extensibility element -->*
            <wsdl:input>?
                <wsdl:documentation ... />?
                <-- extensibility element -->
            </wsdl:input>
            <wsdl:output>?
                <wsdl:documentation ... />?
                <-- extensibility element -->*
            </wsdl:output>
            <wsdl:fault name="nmtoken">?
                <wsdl:documentation ... />?
                <-- extensibility element -->*
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>

    <wsdl:service name="nmtoken">*
        <wsdl:documentation ... />?
        <wsdl:port name="nmtoken" binding="qname">*
            <wsdl:documentation ... />?
            <-- extensibility element -->
        </wsdl:port>
        <-- extensibility element -->
    </wsdl:service>

    <-- extensibility element -->*

    </wsdl:definitions>
```

WSDL 文档的根元素是<definitions>，在根元素之下包含了若干子元素，这些子元素包括以下一些。

（1）types（类型）：数据类型定义的容器，提供了用于描述正在交换的消息的数据类型定义，一般使用 XML Schema 中的类型系统。

（2）message（消息）：通信消息数据结构的抽象定义。message 使用 types 所定义的类型来定义整个消息的数据结构。

（3）operation（操作）：对服务中所支持的操作的抽象描述，一般单个 operation 描述了一对访问入口的请求/响应消息。

（4）porttype（端口类型）：描述了一组操作，每个操作指向一个输入消息和多个输

出消息规范。

（5）binding（绑定）：为特定端口类型定义的操作和消息制定具体的协议和数据格式。

（6）port（端口）：指定用于绑定的地址，定义服务访问点。

（7）service：相关服务访问点的集合。

各个元素之间的逻辑关系如图 4-9 所示。

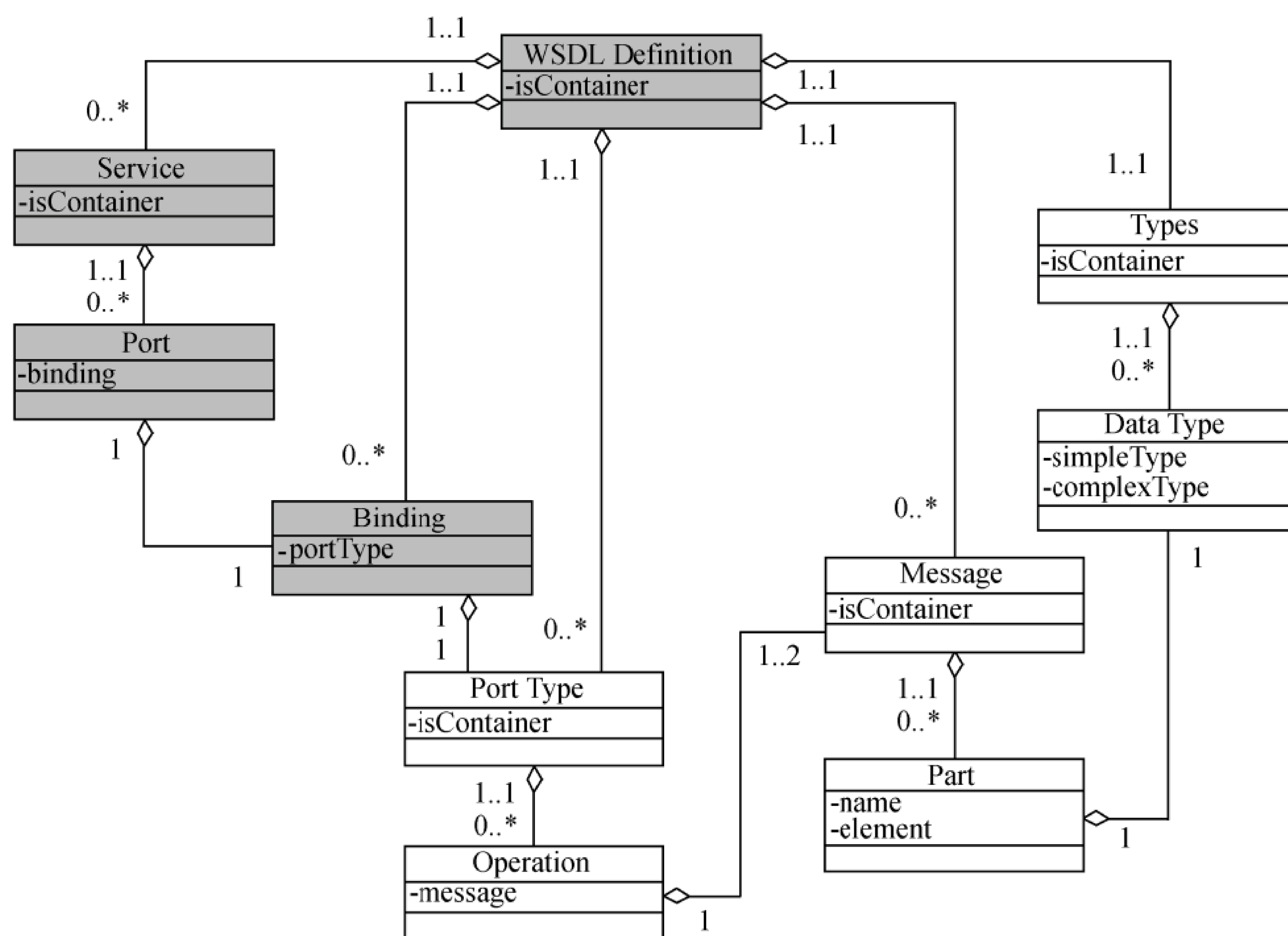


图 4-9 逻辑关系图

下面简单介绍各元素的作用与书写规范。

1．definitions

<definitions>元素是 WSDL 文档的根元素，任何 WSDL 文档有且仅有一个<definitions>元素。在<definitions>元素中可以定义全局的命名空间（namespace）。

2．import

<import>元素紧随<definitions>之后出现，<import>元素与 C＋＋中的#include 的功能类似，可以引用其他的文档。通过<import>元素，用户可以把整个 WSDL 文档分成独立的若干部分，然后通过<import>连接起来。这对于一个很大的 WSDL 文档来说，既便于编写也便于维护。

3．types

<types>元素是一个容器类型的元素，在<types>元素中可以定义一系列的数据类型供<message>使用。目前，XSD（XML Schema Definitions，XML 样式定义）类型系统是最常用的数据类型系统，在<types>元素中既可以沿用 XSD 类型系统，也可以根据需要定义新的类型。例如，下面就定义了一个简单类型——Reference：

```
<xsd:simpleType name="Reference">
    <xsd:restriction base="xsd:string" />
</xsd:simpleType>
```

4．message

<message>元素用来为数据交换建立模型，在<message>元素中将引用在<types>中定义的数据类型。一个<message>元素包含一个或多个<part>元素。一个<part>元素定义一个独立的数据片，这个数据片将是整个消息的一部分。每个<part>元素都与某种类型系统中的类型相关。例如：

```
<message name="SayHelloRequest">
    <part name="firstName" type="xsd:string"/>
</message>
```

在这个例子中，首先通过 name 属性定义了一条消息：SayHelloRequest，这条消息中包含一个 xsd:string 类型的数据——firstName。

5．portType

<portType>元素定义了一组抽象操作和涉及到的抽象消息，一个<portType>元素可以包含一组<operation>元素，例如：

```
<portType name="Hello_PortType">
    <operation name="sayHello">
        <input message="tns:SayHelloRequest"/>
        <output message="tns:SayHelloResponse"/>
    </operation>
</portType>
```

在 WSDL 规范中定义了 4 种交换原语，分别是单向、请求/相应、恳请/响应、通知。

（1）单向（one-way）。单向操作指的是 Web Service 客户端向服务器发送一条不要求服务器回应的消息，即只有 input 消息，而没有 output 消息。其语法如下：

```
<wsdl:operation name="nmtoken">
    <wsdl:input name="nmtoken"? message="qname"/>
</wsdl:operation>
```

（2）请求/响应（request–response）。请求/响应操作指的是客户端向服务器发送一条请求，期望服务器同步的返回该请求的响应，这种操作模式非常类似于过程调用，在服务器没有返回响应前，客户端会一直阻塞。其语法如下：

```
<wsdl:operation name="nmtoken">
    <wsdl:input name="nmtoken"? message="qname"/>
    <wsdl:output name="nmtoken"? message="qname"/>
    <wsdl:fault name="nmtoken"? message="qname"/>*
</wsdl:operation>
```

需要注意的是，在请求/响应操作中，input 消息必须在 output 消息之前出现。

（3）恳求/响应（Solicit Response）。恳求/响应操作虽然在名字上与请求/响应操作类似，但其实完全不一样。恳求/响应操作指的是服务器向客户端恳求得到客户端的响应。换句话说，服务器首先向客户端发送一条请求，客户端在接收到请求之后给予响应。其语法如下：

```
<wsdl:operation name="nmtoken">
    <wsdl:output name="nmtoken"? message="qname"/>
    <wsdl:input name="nmtoken"? message="qname"/>
    <wsdl:fault name="nmtoken"? message="qname"/>*
</wsdl:operation>
```

从中可以看出，恳求/响应操作与请求/响应操作互逆，首先出现的消息是 output，然后才是客户端给服务器的 input 消息。与请求/响应操作一样，这里面的 input 和 output 的顺序也不能颠倒。

（4）通知（notification）。通知操作是一种与单向操作互逆的操作，指的是 Web Service 服务器向客户端发送一条不需要回应的消息。其语法如下：

```
<wsdl:operation name="nmtoken">
    <wsdl:output name="nmtoken"? message="qname"/>
</wsdl:operation>
```

从中可以看出，通知操作仅有 output 消息，而没有来自客户端的 input 消息。

6．binding

<binding>元素是对于特定的<portType>元素定义的具体的协议和数据格式，对于一个给定的<portType>可以有任意数目的绑定。通过<binding>元素中指定的协议可以确定具体的绑定方式（SOAP 绑定、HTTP 绑定或 MIME 绑定）。同时，也可以由指定的协议确定具体的数据格式（例如，HTTP 协议对应了 header/body 的数据格式）。绑定是调用 Web Service 的一个重要操作，不同的绑定方式有各自的特点，其中以 SOAP 绑定最为常用。

7．service

虽然在<binding>元素中非常详细的指明了 Web Service 的访问方法（协议和数据格式）但并没有给出访问 Web Service 的实际 URL。在<service>元素中将给出访问 Web Service 的 URL 并完成整个 Web Service 的描述。在<service>元素中可以包含一个或多个<port>元素，每一个<port>元素都表明了一个服务访问点，如下所示：

```
<service name="Hello_Service">
    <documentation>WSDL File for HelloService</documentation>
    <port binding="tns:Hello_Binding" name="Hello_Port">
        <soap:address location="http://localhost:8080/soap/servlet/
        rpcrouter"/>
    </port>
</service>
```

希赛教育专家提示：一些 WSDL 文档中根本就没有<service>元素。这是因为，WSDL 文档的目标是描述 Web Service 的抽象接口，而<service>元素描述的是 Web 服务访问点。

下面，给出 Hello CSAI 的 WSDL 文档的完整内容供读者参考，自动生成该文档的方法将在 4.5.1 节中介绍。

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap=http://schemas.
    xmlsoap.org/wsdl/soap/ xmlns:s=http://www.w3.org/2001/XMLSchema
    xmlns:s0="http://tempuri.org/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="http://tempuri.org/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
    <s:schemaelementFormDefault="qualified"
        targetNamespace="http://tempuri.org/">
    <s:element name="SayHello">
    <s:complexType>
        <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="name" type="s:
        string" />
        </s:sequence>
    </s:complexType>
    </s:element>
```

```
  <s:element name="SayHelloResponse">
     <s:complexType>
        <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="SayHelloResult"
        type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:element>
  <s:element name="string" nillable="true" type="s:string" />
  </s:schema>
</types>
<message name="SayHelloSoapIn">
   <part name="parameters" element="s0:SayHello" />
</message>
<message name="SayHelloSoapOut">
  <part name="parameters" element="s0:SayHelloResponse" />
</message>
<message name="SayHelloHttpGetIn">
   <part name="name" type="s:string" />
</message>
<message name="SayHelloHttpGetOut">
   <part name="Body" element="s0:string" />
</message>
<message name="SayHelloHttpPostIn">
  <part name="name" type="s:string" />
</message>
<message name="SayHelloHttpPostOut">
   <part name="Body" element="s0:string" />
</message>
<portType name="HelloMessageSoap">
   <operation name="SayHello">
      <input message="s0:SayHelloSoapIn" />
      <output message="s0:SayHelloSoapOut" />
   </operation>
</portType>
<portType name="HelloMessageHttpGet">
   <operation name="SayHello">
      <input message="s0:SayHelloHttpGetIn" />
      <output message="s0:SayHelloHttpGetOut" />
   </operation>
</portType>
```

```
<portType name="HelloMessageHttpPost">
  <operation name="SayHello">
        <input message="s0:SayHelloHttpPostIn" />
        <output message="s0:SayHelloHttpPostOut" />
   </operation>
</portType>
<binding name="HelloMessageSoap" type="s0:HelloMessageSoap">
    <soap:binding transport=
        "http://schemas.xmlsoap.org/soap/http" style="document" />
<operation name="SayHello">
    <soap:operation soapAction="http://tempuri.org/SayHello"
           style="document" />
        <input>
            <soap:body use="literal" />
        </input>
        <output>
            <soap:body use="literal" />
    </output>
</operation>
</binding>
<binding name="HelloMessageHttpGet" type="s0:HelloMessageHttpGet">
    <http:binding verb="GET" />
    <operation name="SayHello">
        <http:operation location="/SayHello" />
           <input>
             <http:urlEncoded />
           </input>
           <output>
                 <mime:mimeXml part="Body" />
           </output>
    </operation>
</binding>
<binding name="HelloMessageHttpPost" type="s0:HelloMessageHttpPost">
  <http:binding verb="POST" />
  <operation name="SayHello">
    <http:operation location="/SayHello" />
    <input>
      <mime:content type="application/x-www-form-urlencoded" />
    </input>
    <output>
      <mime:mimeXml part="Body" />
```

```
      </output>
    </operation>
  </binding>
  <service name="HelloMessage">
    <port name="HelloMessageSoap" binding="s0:HelloMessageSoap">
      <soap:address location="http://localhost/HelloService.asmx" />
    </port>
    <port name="HelloMessageHttpGet" binding="s0:HelloMessageHttpGet">
      <http:address location="http://localhost/HelloService.asmx" />
    </port>
    <port name="HelloMessageHttpPost" binding="s0:HelloMessageHttpPost">
      <http:address location="http://localhost/HelloService.asmx" />
    </port>
  </service>
</definitions>
```

4.3 UDDI

UDDI 是一种用于描述、发现、集成 Web Service 的技术，它是 Web Service 协议栈的一个重要部分。通过 UDDI，企业可以根据自己的需要动态查找并使用 Web 服务，也可以将自己的 Web 服务动态地发布到 UDDI 注册中心，供其他用户使用。

UDDI 对于 B2B 模式的电子商务有着巨大的作用。例如，A 公司是一个半导体配件的制造商，通过将自己的 Web 服务发布到 UDDI 注册中心，A 公司的产品可以很容易地被分布于世界各地的采购方查找到，从而加入到全球的供应链中。同时，B 公司是一个半导体配件的采购方，他既从 A 公司采购部分半导体配件，也从其他的若干家公司进行原料的采购。使用 UDDI 和 Web Service，B 公司可以更容易地将他和原材料供应厂商的系统集成在一起。那么 B 公司可以使用统一的平台以统一的方式管理他的供应链，可以通过自己的信息系统和互联网，直接在其他公司的 Web Service 平台上下电子订单。当然，B 公司也可以使用 UDDI 来查找个合适的供货商，并迅速地把这个新的合作伙伴加入到自己的子系统中。

在 UDDI 技术规范中，主要包含以下三个部分的内容：

（1）UDDI 数据模型。UDDI 数据模型是一个用于描述商业组织和 Web Service 的 XML Schema。

（2）UDDI API。UDDI API 是一组用于查找或发布 UDDI 数据的方法，UDDI API 基于 SOAP。

（3）UDDI 注册服务。UDDI 注册服务数据是 Web Service 中的一种基础设施，UDDI 注册服务对应着服务注册中心的角色。

4.3.1　UDDI 数据模型

在 UDDI 中定义了 4 种基本的数据结构，分别是 businessEntity、businessService、bindingTemplate 和 tModel，这 4 种基本数据结构的关系如图 4-10 所示。

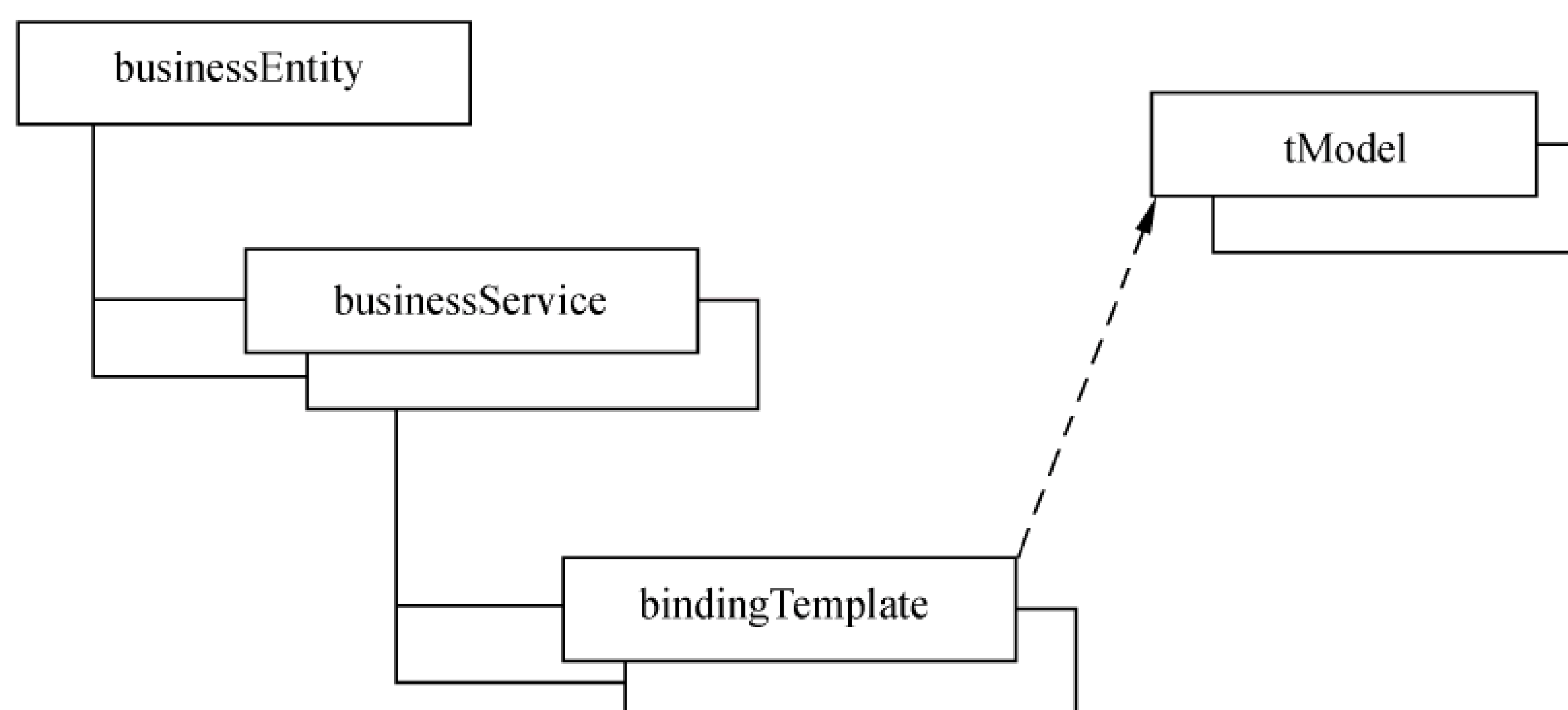


图 4-10　基本数据结构关系

1．businessEntity

businessEntity 元素对企业信息进行描述，包括企业的基本信息（企业名称、联系方式等）、分类信息（企业的类型等）以及标识信息。例如，下面是 Microsoft 的 businessEntity 中描述企业基本信息的一部分：

```
<businessEntity businessKey="0076b468-eb27-42e5-ac09-9955cff462a3"
    operator="Microsoft Corporation" authorizedName="Martin Kohlleppel">
    <name>Microsoft Corporation</name>
    <description xml:lang="en">Empowering people through great software
     any time, any place and on any device is Microsoft's vision. As the
     worldwide leader in software for personal and business computing, we
     strive to produce innovative products and services that meet our
     customer's...
    </description>
    <contacts>
      <contact useType="Corporate Addresses and telephone">
        <description xml:lang="en">Corporate Mailing
            Addresses</description>
        <personName />
        <phone useType="Corporate Headquarters">(425) 882-8080</phone>
        <address sortCode="~" useType="Corporate Headquarters">
          <addressLine>Microsoft Corporation</addressLine>
          <addressLine>One Microsoft Way</addressLine>
```

```
            <addressLine>Redmond, WA 98052-6399</addressLine>
            <addressLine>USA</addressLine>
          </address>
        </contact>
        <contact useType="Technical Contact - Corporate UD">
        <description xml:lang="en">World Wide Operations</description>
          <personName>Martin Kohlleppel</personName>
          <email>martink@microsoft.com</email>
        </contact>
      </contacts>
      <identifierBag>
        <keyedReference tModelKey="uuid:8609c81e-ee1f-4d5a-b202-3eb13ad0
        1823"
          keyName="D-U-N-S" keyValue="08-146-6849" />
      </identifierBag>
      <categoryBag>
        <keyedReference tModelKey="uuid:c0b9fe13-179f-413d-8a5b-5004db8e5
        bb2"
          keyName="NAICS: Software Publisher" keyValue="51121" />
      </categoryBag>
    </businessEntity>
```

从这个例子中可以看出，在 businessEntity 中包含了商业实体中的基本信息。其中第一行中的 businessKey 属性是一个唯一的键值，这个唯一的标识将用来发布服务。除了基本的联系信息外，这一段描述还在<identifierBag>和<categoryBag>中描述了商业标识和商业分类。使用 XML Schema 对 businessEntity 的定义如下：

```
<element name="businessEntity">
  <complexType>
    <sequence>
      <element ref="discoveryURLs" minOccurs="0"/>
      <element ref="name" maxOccurs="unbounded"/>
      <element ref="description" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="contacts" minOccurs="0"/>
      <element ref="bussinessService" minOccurs="0"/>
      <element ref="identifierBag" minOccurs="0"/>
      <element ref="categoryBag" minOccurs="0"/>
    </sequence>
    <attribute ref="businessKey" use="required"/>
    <attribute ref="operator"/>
    <attribute ref="authorizedName">
```

```
  </complexType>
</element>
```

2．businessService

如上面的 businessEntity 定义所示，在 businessEntity 中可以包含一组 businessService 元素。businessService 描述了一个单一的或一组相关的 Web Service，其中包括名称、描述和一组 bindingTemplate。businessService 结构的定义如下：

```
<element name="businessService">
  <complexType>
    <sequence>
      <element ref="name" maxOccurs="unbounded"/>
      <element ref="description" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="bindingTemplates"/>
      <element ref="categoryBag" minOccurs="0"/>
    </sequence>
    <attribute ref="serviceKey" use="required"/>
    <attribute ref="businessKey"/>
  </complexType>
</element>
```

3．bindingTemplate

如图 4-10 所示，一组 bindingTemplate 元素包含在 businessService 中。在 bindingTemplate 中描述了如何访问一个特定的 Web Service 并给出了访问点的地址（accessPoint）。bindingTemplate 的结构定义如下：

```
<element name="bindingTemplate">
  <complexType>
    <sequence>
      <element ref="description" minOccurs="0" maxOccurs="unbounded"/>
      <choice>
        <element ref="accessPoint" minOccurs="0"/>
        <element ref="hostingRedirector" minOccurs="0"/>
      </choice>
      <element ref="tModelInstanceDetails"/>
    </sequence>
    <attribute ref="bindingKey" use="required"/>
    <attribute ref="serviceKey"/>
  </complexType>
</element>
```

下面，给出 XMethod.net 的 Web 服务中对于 businessService 和 bindingTemplate 描述的片断，以给读者以直观的印象：

```
<businessService
  serviceKey="d5921160-3e16-11d5-98bf-002035229c64"
  businessKey="ba744ed0-3aaf-11d5-80dc-002035229c64">
  <name>XMethods Delayed Stock Quotes</name>
  <description xml:lang="en">20-minute delayed stock quotes</description>
  <bindingTemplates>
    <bindingTemplate
      serviceKey="d5921160-3e16-11d5-98bf-002035229c64"
      bindingKey="d594a970-3e16-11d5-98bf-002035229c64">
      <description xml:lang="en">
        SOAP binding for delayed stock quotes service
      </description>
      <accessPoint URLType="http">
        http://services.xmethods.net:80/soap
      </accessPoint>
      <tModelInstanceDetails>
        <tModelInstanceInfo
          tModelKey="uuid:0e727db0-3e14-11d5-98bf-002035229c64" />
      </tModelInstanceDetails>
    </bindingTemplate>
  </bindingTemplates>
</businessService>
```

4．tModel

在 bindingTemplate 元素中包含了一个服务的 URL 和一个 tModel。tModel 结构是 UDDI 数据模型中最复杂也是最难理解的结构。tModel 是描述技术规范的模型，它可以为外部的技术规范提供引用地址。例如，在上面 XMethods.net 的例子中，虽然 bindingTemplate 提供了访问 Web Service 的相关信息（如 SOAP 绑定的地址等），但是在 bindingTemplate 中并没有指定这个 Web Service 的接口，也就是说客户仍然无法直接访问该 Web Service。而 tModel 通过对外部规范的引用，可以解决这个问题。看下面的例子：

```
<tModel
  tModelKey="uuid:0e727db0-3e14-11d5-98bf-002035229c64"
  operator="www.ibm.com/services/uddi" authorizedName="0100001QS1">
  <name>XMethods Simple Stock Quote</name>
  <description xml:lang="en">Simple stock quote interface</description>
```

```
    <overviewDoc>
      <description xml:lang="en">wsdl link</description>
      <overviewURL>
        http://www.xmethods.net/tmodels/SimpleStockQuote.wsdl
      </overviewURL>
    </overviewDoc>
    <categoryBag>
      <keyedReference
        tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"
          keyName="uddi-org:types" keyValue="wsdlSpec" />
    </categoryBag>
  </tModel>
```

这个例子在<overviewDoc>中给出了这个 Web Service 接口的技术规范（一份 WSDL 文档）和获得该技术规范的地址（http://www.xmethods.net/tmodels/SimpleStockQuote.wsdl）tModel 结构的定义如下：

```
<element name="tModel">
  <complexType>
    <sequence>
      <element ref="name"/>
      <element ref="description" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="overviewDoc" minOccurs="0"/>
      <element ref="identifierBag" minOccurs="0"/>
      <element ref="categoryBag" minOccurs="0"/>
    </sequence>
    <attribute ref="tModelKey" use="required"/>
    <attribute ref="operator"/>
    <attribute ref="authorizedName"/>
  </complexType>
</element>
```

4.3.2 注册 Web 服务

使用 UDDI 注册 Web Service 可分为两个步骤。首先，为 UDDI 条目建立模型；然后，将 UDDI 条目注册到 UDDI 注册中心。建立 UDDI 条目时，需要确定以下事项：

（1）确定 Web Service 的 tModel（WSDL 文档）。

（2）确定组织（企业、事业单位、各类机构等，下同）名称、简介以及所提供的 Web Service 的主要联系方法。

（3）确定组织正确的标识和分类。

（4）确定组织通过 UDDI 提供的 Web Service。

（5）确定所提供的 Web Service 的正确分类。

事实上，这一系列的步骤也正是为了根据 UDDI 指定的数据结构建立正确的 UDDI 条目。在完成 UDDI 条目建模之后，就可以将其发布到 UDDI 注册中心。通过 UDDI 注册中心的 Web 页面和使用 UDDI API 编写程序都可以完成 Web Service 的发布。使用 UDDI API 可以直接编写程序，把 Web Service 发布到 UDDI 注册中心的服务器上，由于篇幅限制，本书将不再介绍 UDDI API 的详细内容，感兴趣的读者可以查阅参考文献中的相关资料。

4.3.3 调用 Web 服务

使用 UDDI 调用 Web Service 的步骤如下：

（1）编写调用 Web Service 的程序时，程序员使用 UDDI 注册中心来定位并获得 businessEntity。

（2）程序员可以进一步获得更详细的 businessService 信息，或是得到一个完整的 businessEntity 结构。因为 businessEntity 结构中包含了已发布的 Web Service 的全部信息，所以，程序员可以从中选择一个 bindingTemplate 待以后使用。

（3）Web Service 在 tModel 中提供了相关规范的引用地址，程序员可以根据引用地址获得规范并编写程序。

（4）运行时，程序员可以按需要使用已经保存下来的 bindingTemplate 信息，调用 Web Service。

4.4 SOAP

SOAP 以 XML 形式提供一个简单、轻量的用于在分散或分布环境中交换结构化和类型信息的机制。SOAP 本身并没有定义任何应用程序语义（如编程模型或特定语义的实现），实际上，它通过提供一个有标准构件的包模型和在模块中编码数据的机制，定义了一个简单的表示应用程序语义的机制。这使 SOAP 能够被用于从消息传递到 RPC 的各种系统。

SOAP 主要包括以下 4 个部分：

（1）SOAP 封装结构。定义一个整体框架用来表示消息中包含什么内容，谁来处理这些内容，以及这些内容是可选的或是必需的。

（2）SOAP 编码规则。用以交换应用程序定义的数据类型的实例的一系列机制。

（3）SOAP RPC 表示。定义一个用来表示远程过程调用和应答的协议。

（4）SOAP 绑定。定义一个使用底层传输协议来完成在节点间交换 SOAP 信封的约定。

SOAP 客户端是一种能够创建 XML 文档的本地应用程序，它不仅可以是普通的桌面应用程序，还可以是一种基于 Web 的应用程序，如 Web 表单等。它所创建的 XML 文档应该包含在分布式RPC所需的信息，这些消息和请求一般都是通过HTTP进行传递的，而通常的安全机制都不会禁止 HTTP 协议，因此，也就意味着 SOAP 协议可以顺利地通过防火墙，实现跨越不同平台的信息交换。

SOAP 服务器通常只是用来监听 SOAP 消息的特殊代码，它可以对 SOAP 文档进行分配和解释。它通常可以与基于 J2EE 技术或是.NET 技术的应用程序服务器交互，这种应用程序服务器可以处理多种客户端的 SOAP 请求。SOAP 服务器也是通过 HTTP 连接接收文档，然后将其转换成为可以被另一端对象理解的语言。由于 SOAP 在所有的通信中都采用 XML 格式，因此，可以用来实现两种不同语言中的对象之间的通信。

从某种意义上，可以将 SOAP 简单地理解为：SOAP= HTTP+RPC+XML，也就是采用 HTTP 作为底层通信协议，RPC 作为通用的调用途径，XML 作为数据打包的格式，提供了一个能够穿越防火墙的通信交互能力。

4.4.1　消息封装和编码规则

SOAP 消息是一个 XML 文档，在实际应用中，提到 SOAP 消息时，往往就是指这个 XML 文档。那么，SOAP 是如何进行封装的，如何进行编码的呢？这就是本节要介绍的内容。

1．SOAP 的消息封装

SOAP 消息包括以下三个部分：

（1）封装。封装的元素名是“Envelope”，在表示消息的 XML 文档中，封装是顶层元素，在 SOAP 消息中必须出现。

（2）SOAP 头。SOAP 头的元素名是“Header”，提供了向 SOAP 消息中添加关于这条 SOAP 消息的某些要素（feature）的机制。SOAP 定义了少量的属性用来表明这项要素是否可选以及由谁来处理。SOAP 头在 SOAP 消息中可能出现，也可能不出现。如果出现的话，必须是 SOAP 封装元素的第一个直接子元素。SOAP 头可以包含多个条目，每个条目都是 SOAP 头元素的直接子元素。

（3）SOAP 体。SOAP 体的元素名是“Body”，是包含消息的最终接收者想要的信息的容器。SOAP 体在 SOAP 消息中必须出现且必须是 SOAP 封装元素的直接子元素。如果有头元素，则 SOAP 体必须直接跟在 SOAP 头元素之后；如果没有头元素，则 SOAP 体必须是 SOAP 封装元素的第一个直接子元素。SOAP 体可以包括多个条目，每个条目必须是 SOAP 体元素的直接子元素。

2．SOAP 的编码规则

XML 允许非常灵活的数据编码方式，SOAP 定义了一个较小的规则集合，下面的术语用来描述编码规则。

（1）值（value）。值是一个字符串、类型（数字、日期、枚举等）的名或是几个简单值的组合。所有的值都有特定的类型。

（2）简单值（Simple Value）。简单值没有名部分，如特定的字符串、整数、枚举值等。

（3）复合值（Compound Value）。复合值是相关的值的组合，如订单、股票报表、街道地址等。在复合值中，每个相关的值都以名、序号或这两者来区分，这称为访问者（accessor）。在复合值中，多个访问者有相同的名是允许的。

（4）数组（array）。数组是一个复合值，成员值按照在数组中的位置相互区分。

（5）结构（struct）。结构也是一个复合值，成员值之间的唯一区别是访问者的名字，访问者名互不相同。

（6）简单类型（Simple Type）。简单类型是简单值的类，如字符串类、整数类、枚举类等。

（7）复合类型（Compound Type）。复合类型是复合值的类，它们有相同的访问者名，但可能会有不同的值。

在复合类型中，如果类型内的访问者名互不相同，但是可能与其他类型中的访问者名相同，即访问者名加上类型名形成一个唯一的标志符，这个名叫作“局部范围名”。如果名是直接或间接的基于通用资源标志符（Universal Resource Identifier，URI）的一部分，那么不管它出现在什么类型中，这个名本身就可以唯一标志这个访问者，这样的名叫作“全局范围名”。

所有的值以元素内容的形式表示，对于每个具有值的元素，值的类型必须用下述三种方式之一描述：

（1）所属元素实例有 xsi:type 属性。

（2）所属元素是一个有 SOAP-ENC:arrayType 属性（该属性可能是缺省的）的元素的子元素。

（3）所属元素的名具有特定的类型，类型可以由 schema 确定。

4.4.2 SOAP 应用

SOAP 的应用很广泛，本节简单介绍 SOAP 在 HTTP 和 RPC 中的应用。

1．在 HTTP 中使用 SOAP

把 SOAP 绑定到 HTTP，无论使用或不用 HTTP 扩展框架，都有很大的好处：在利用 SOAP 的形式化和灵活性的同时，使用 HTTP 种种丰富的特性。在 HTTP 中携带 SOAP 消息，并不意味着 SOAP 改写了 HTTP 已有的语义，而是将构建在 HTTP 之上 SOAP 语义自然地对应到 HTTP 语义。SOAP 自然地遵循 HTTP 的请求/应答消息模型，使得 SOAP 的请求和应答参数可以包含在 HTTP 请求和应答中。

希赛教育专家提示：SOAP 的中间节点与 HTTP 的中间节点并不等同，也就是说，

不要期望一个根据 HTTP 连接头中的域寻址到的 HTTP 中间节点能够检查或处理 HTTP 请求中的 SOAP 消息。在 HTTP 消息中包含 SOAP 消息时，应用程序必须使用媒体类型 text/xml。

（1）SOAP HTTP 请求

虽然 SOAP 可能与各种 HTTP 请求方式相结合，但是绑定仅定义了在 HTTP POST 请求中包含 SOAP 消息。

（2）HTTP 头中 SOAPAction 域

一个 HTTP 请求头中的 SOAPAction 域用来指出这是一个 SOAP HTTP 请求，它的值是所要的 URI。在格式、URI 的特性和可解析性上没有任何限制。当 HTTP 客户发出 SOAP HTTP 请求时必须使用在 HTTP 头中使用这个域。

```
soapaction = "SOAPAction" ":" [ <"> URI-reference <"> ]
URI-reference = <as defined in RFC 2396 [4]>
```

HTTP 头中的 SOAPAction 域使服务器能正确的过滤 HTTP 中 SOAP 请求消息。如果这个域的值是空字符串（""），表示 SOAP 消息的目标就是 HTTP 请求的 URI。这个域没有值，则表示没有 SOAP 消息的目标的信息。例如：

```
SOAPAction: "http://electrocommerce.org/abc#MyMessage"
SOAPAction: "myapp.sdl"
SOAPAction: ""
SOAPAction:
```

（3）SOAP HTTP 应答

SOAP HTTP 遵循 HTTP 中表示通信状态信息的 HTTP 状态码的语义。例如，2xx 状态码表示这个包含了 SOAP 构件的客户请求已经被成功的收到、理解和接收。在处理请求时如果发生错误，SOAP HTTP 服务器必须发出应答 HTTP 500“Internal Server Error”，并在这个应答中包含一个 SOAP Fault 元素，表示这个 SOAP 处理错误。

（4）HTTP 扩展框架

一个 SOAP 消息可以与 HTTP 扩展框架一起使用以区分是否有 SOAP HTTP 请求和它的目标。是使用扩展框架或是普通的 HTTP，关系到通信各方的策略和能力。通过使用一个必需的扩展声明和“M-”HTTP 方法名前缀，客户可以强制使用 HTTP 扩展框架。服务器可以使用 HTTP 状态码 510 “Not Extended”强制使用 HTTP 扩展框架。也就是说，使用一个额外的消息，任何一方都可以发现另一方的策略并依照执行。

（5）SOAP HTTP 举例

使用 POST 的 SOAP HTTP：

```
POST /StockQuote HTTP/1.1
```

```
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://electrocommerce.org/abc#MyMessage"
<SOAP-ENV:Envelope... HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope...
```

使用扩展框架的 SOAP HTTP：

```
M-POST /StockQuote HTTP/1.1 Man:
"http://schemas.xmlsoap.org/soap/envelope/"; ns=NNNN
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
NNNN-SOAPAction:
"http://electrocommerce.org/abc#MyMessage"
<SOAP-ENV:Envelope...
HTTP/1.1 200 OK
Ext:
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope...
```

2．在 RPC 中使用 SOAP

设计 SOAP 的目的之一就是利用 XML 的扩展性和灵活性来封装和交换 RPC 调用，在 RPC 中使用 SOAP 和 SOAP 协议绑定是紧密相关的。在使用 HTTP 作为绑定协议时，一个 RPC 调用自然地映射到一个 HTTP 请求，RPC 应答同样映射到 HTTP 应答。但是，在 RPC 中使用 SOAP 并不限于绑定 HTTP 协议。

要进行方法调用，一般需要以下信息：目标对象的 URI、方法名、方法签名（signature）、方法的参数、头数据，其中方法签名和头数据是可选的。SOAP 依靠协议绑定提供传送 URI 的机制。例如，对 HTTP 来说，请求的 URI 指出了调用的来源。除了必须是一个合法的 URI 之外，SOAP 对一个地址的格式没有任何限制。

1）RPC 和 SOAP 体

RPC 方法调用和应答都包含在 SOAP Body 元素中，它们使用如下的表示形式：

- 方法调用用结构表示。一个方法调用被看作单个的结构，每个[in]和[in/out]参数有一个访问者。结构的名和类型与方法相同，它们的出现顺序和方法中定义的参数顺序相同。
- 方法应答用结构表示。一个方法应答被看作单个的结构，返回值和每个[in]和

[in/out]参数有一个访问者。第一个访问者是返回值，之后是参数访问者，参数访问者的出现顺序和方法中定义的参数顺序相同。每个参数访问者的名称和类型与参数的名称和类型相对应。

- 方法错误用 SOAP Fault 元素表示。如果绑定的协议有额外的规则表示错误，则这些规则也必须要遵从。因为返回结果表示调用成功，错误表示调用失败，所以，如果在方法应答中同时包含“返回结果”和“错误表示”，则是错误的。

2）RPC 和 SOAP 头

在 RPC 编码中，可能会有与方法请求有关但不是正规的方法签名的附加信息。如果这样，它必须作为 SOAP 头元素的子元素。使用这种头元素的一个例子是在消息中传递事务 ID。由于事务 ID 不是方法签名的一部分，通常由底层的构件而不是应用程序代码控制，所以没有一种直接的方法在调用中传递这个必要的信息。通过在 SOAP 头中添加一个给定名字的条目，接收方的事务管理器就可以析取这个事务 ID，而且不影响 RPC 的代码。

3）示例

一个消息处理器从传入消息中检索并除去 SOAP 头的示例代码如下：

```
public class SOAPHeaderHandler extends javax.xml.rpc.handler.Generic-
Handler {
    ...
    public boolean handleRequest(MessageContext arg) {
     if (arg instanceof SOAPMessageContext) {
        SOAPMessageContext context = (SOAPMessageContext)arg;
        try {
            SOAPHeader header =
            context.getMessage().getSOAPPart().getEnvelope().getHeader();
            Iterator headers =
                    header.extractHeaderElements
                    ("http://schemas.xmlsoap.org/soap/actor/next");
            while (headers.hasNext()) {
                SOAPHeaderElement he =
                            (SOAPHeaderElement)headers.next();
                // process the retrieved header element here
            }
        } catch (SOAPException x) {
            // insert error handling here
        }
    }
    return true;
```

```
    }
}
```

4.5 构造一个简单的 Web Service

作为一种采用松散耦合结构集成各式各样应用程序的标准，Web Service 的技术特色在于动态发布、描述和调用机制。换句话说，Web Service 的技术闪光点正在于 SOAP、UDDI 和 WSDL 这三种协议巧妙的结合。相比之下，书写一段 Web Service 程序则显得轻松得多，并不比编写普通的应用程序更复杂。本节给出使用.NET 平台开发 Web Service 的例程供读者参考。

4.5.1 编写服务器端

使用.NET 平台开发 Web Service 的系统要求：Windows 2000 Server 或以上版本，安装有.NET Frameworks SDK 和 IIS（Internet Information Server）5.0 以上版本。使用任何一个文本编辑器（如记事本等）编写如下代码：

```
<@ WebService Language="C#" Class="HelloMessage"%>
using System;
using System.Web.Services;
public class HelloMessage:WebService
{
    [WebMethod] public String SayHello(String name)
    {
        return "Hello, "+name;
    }
}
```

将这段代码保存为 HelloService.asmx，然后把这个文件复制到 IIS 服务的根目录下。打开浏览器，访问地址http://localhost/HelloService.asmx 就可以看到该 Web Service 的测试页面。在这个测试页面中列出了 Web Service 提供的访问接口，对于编写的 Web Service 来说，将会列出 SayHello 方法。单击该方法的链接将会进入该方法的调试页面。在 SayHello 方法中包含一个参数：String name，在调试页面中有一个文本框用于输入该参数。在参数输入框中输入 CSAI，并单击“调用”按钮，IIS 将会返回一个 XML 文件，内容如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<String xmlns="http://tempuri.org/">Hello, CSAI</string>
```

其中包含了该方法的调用结果和结果类型。下面，对这段程序进行简单的说明。

```
<@ WebService Language="C#" Class="HelloMessage"%>
```

这一行代码声明了该程序是一个使用 C#语言编写的 Web Service 程序。

```
using System;
using System.Web.Services;
```

这两行代码引用了命名空间，第一个是几乎所有 C#语言都会使用到的 System，另一个是 Web Service 程序中使用到的 System.Web.Services。

完成声明之后，程序定义了类 HelloMessage，类 HelloMessage 继承自 WebService。类中定义了一个 Web Service 方法：public String SayHello(String name)，为了标识该方法可以通过 Internet 调用，在方法前使用了[WebMethod]作为修饰。所有的 Web Service 方法都必须有这个修饰，否则不能被服务器处理。该方法只有一行代码，返回了字符串 Hello 和参数字符串的合成。

紧接着，可以生成该服务的 WSDL 文档，调用 http://localhost/HelloService.asmx?WSDL，服务器将自动生成描述文档并返回。在 4.2.3 节的最后所给出的文档就是由服务器自动生成的 WSDL 文档。

4.5.2　编写客户端

前面提到过，调用 Web Service 可以通过静态绑定和动态绑定两种方式，在本节中，使用静态绑定的方式调用 4.5.1 节编写的 Web Service。

首先，需要根据 Web Service 的描述文档产生相应的服务代理类，在执行过程中，客户使用代理类中的信息访问 Web Service，并进行实际的方法调用。

假设.NET Framework SDK 安装在 C 盘的默认路径下。首先，将 C:\Program Files\Microsoft Visual Studio. NET\FrameworkSDK\Bin 增加到系统的查找路径中（这里的路径仅是一个例子，因安装方式不同会有不同的路径）。在 Framework SDK 中提供了一个自动生成服务代理类的工具 wsdl.exe，这个工具就位于刚才添加的路径中。进入 Windows 2000（或其他版本）命令行方式，输入命令：

```
wsdl.exe /language:c# http://localhost/HelloService.asmx?wsdl
```

稍微等待一小段时间就会看到命令完成的提示，同时生成客户代理类的代码文件 HelloService.cs。然后打开文本编辑器，输入一段代码：

```
using System;
using System.IO;
public class HelloClient
```

```
{
    public static void Main()
    {
        HelloMessage sampClient=new HelloMessage();
        Console.WriteLine(sampClient.SayHello("CSAI"));
    }
}
```

将其保存为HelloClient.cs文件，并将这个文件放在和HelloService.cs相同的目录。

这时，需要使用到另外一个编译工具csc.exe。在默认情况下，csc在c:\WINNT\Microsoft.NET\Framework\V****目录下，其中****代表了Framework SDK的实际版本号，依不同的安装情况而不同。同样，也可以将这个目录放到默认的查找路径中，或直接输入c:\WINNT\Microsoft.NET\Framework\V1.0.3705\csc.exe HelloClient.cs Hello-Message.cs 。这个命令将编译出一个可执行文件HelloClient.exe，这就是调用Web Service的客户端。执行这个文件就可以看到，在系统控制台输出“Hello, CSAI”的结果。

本章参考文献

[1] http://www-900.ibm.com/developerWorks/cn/webservices/
[2] http://www-900.ibm.com/developerWorks/cn/xml/
[3] Carlos C. Tapang.Web Service 描述语言 WSDL 详解. http://www.yesky.com/20011013/200759.shtml
[4] 郑小平. .NET 精髓——Web 服务原理与开发. 北京：人民邮电出版社，2002
[5] W3C. SOAP 协议规范

第 5 章　异构数据库的集成

数据处理作为计算机的主要工作依次经历了文件系统、集中式数据库系统、分布式数据库系统、异构分布式数据库系统等几个不同的发展阶段。传统的数据处理一般以文件方式为主，现代的数据处理方式一般以数据库系统为主。随着应用的不断扩大，传统数据库系统方式已越来越不能满足现代数据处理的要求，同时已有的数据库系统又不可能全部丢弃，异构数据库发展起来就成为必然。

5.1　异构数据库体系结构

异构数据库系统是相关的多个数据库系统的集合，可以实现数据的共享和透明访问。每个数据库系统在加入异构数据库系统之前本身就已经存在，拥有自己的 DBMS，他们在加入异构数据库系统之后仍应具有自治性。在实现数据共享的同时，每个数据库系统仍保留有自己的应用特性、完整性控制和安全性控制。通常，将异构数据库系统中的各个数据库称为参与数据库。参与数据库可以全部在同一场地，也可以分布在多个不同的场地；参与数据库可以是同构的也可以是异构的。

5.1.1　异构性

异构数据库系统的异构性主要体现在以下几个方面。

（1）计算机体系结构的异构：各个参与的数据库可以分别运行在大型机、小型机、工作站、PC 或嵌入式系统中。

（2）基础操作系统的异构：各个数据库系统的基础操作系统可以是 UNIX、Windows Server、Linux 等。

（3）DBMS 本身的异构：可以是同为关系型数据库系统的 Oracle、Microsoft SQL Server 等，也可以是不同数据模型的数据库，如关系、层次、网络、面向对象、函数型数据库共同组成一个异构数据库系统。

异构数据库系统的目标在于实现不同数据库之间的数据信息资源、硬件设备资源和人力资源的合并与共享。其中关键的一点就是以局部数据库模式为基础，建立全局的数据模式或全局外视图。这种全局模式对于建立高级的决策支持系统尤为重要。

一般大型机构在许多地点都有分支机构，每个子机构的数据库中都有着自己的信息数据，而决策制订人员通常只关心宏观的、为全局模式所描述的信息。建立在数据仓库技术基础上的异构数据库全局模式的描述是一种比较好的解决方案。数据仓库可以从异

构数据库系统中的多个数据库中收集信息，并建立统一的全局模式，同时收集的数据还支持对历史数据的访问，用户通过数据仓库提供的统一的数据接口进行决策支持的查询。

5.1.2 数据库转换

异构数据库系统实现数据共享应当达到两点：一是实现数据库转换；二是实现数据的透明访问。

数据库转换是将数据库转换成用户熟悉的等价数据模型，然后根据需要再装入数据，使用户利用自己熟悉的数据库系统和熟悉的查询语言方便地使用以实现数据共享。数据库转换一般首先要进行类型转换，访问源数据库系统，将源数据库的数据定义模型转换为目标数据库的数据定义模型，然后进行数据重组，即将源数据库系统中的数据装入到目标数据库中。

通常，数据库转换需要做以下的工作：

（1）访问源数据库系统。

（2）复制一个源数据库的副本。

（3）把源数据库的副本转换为等价的目标数据库。

（4）把转换好的目标数据库装入到与目标数据库相应的数据库系统。

（5）用目标数据库管理系统中的数据处理语言访问目标数据库。

然而在转换的过程中要实现严格的等价转换往往比较困难，且有时对某一用户是语义等价的，而对另一用户却非语义等价，故在源数据库和目标数据库两种模型之间可能存在各种语法和语义上的冲突，这些冲突可能包括以下几种。

（1）命名冲突：源模型中的标识符可能是目标模型中的保留字，这时就需要重新命名。

（2）格式冲突：同一种数据类型可能有不同的表示方法和语义差异，这时需要定义两种模型之间的变换函数。

（3）结构冲突：如果两种数据库系统之间的数据定义模型不同，如分别为关系模型和层次模型，那么就需要重新定义实体属性和联系，以防止属性或联系信息的丢失。

总之，在进行数据转换后，一方面源数据库模式中所有需要共享的信息都被转换到目标数据库中，另一方面这种转换又不能包含冗余的关联信息。

利用数据库转换工具可以实现不同数据库系统之间的数据模型转换，但要注意的是：如果数据库转换同时进行数据定义模式转换和数据转换，就可能引起同一数据集合在异构数据库系统中存在多个副本，因此需要引入新的访问控制机制。在保证各个参与数据库自治，维护其完整性、安全性的基础上，对异构数据库系统提供全局的访问控制、并发机制和安全控制。

希赛教育专家提示：如果数据库转换只进行数据定义转换，不产生数据的副本，那么在新的目标数据库定义模型的框架下访问数据，实现上仍然是对源数据库系统中数据

的访问。这时，利用新的数据库系统中的数据处理语言实现的事务不能直接访问源数据库，必须进行事务级的翻译才可以执行。

5.1.3　数据的透明访问

经过数据库转换实现的数据共享体现了异构数据的透明访问。在异构数据库系统中实现了数据的透明访问，用户就可以将异构数据库系统看成同构的数据库系统，用自己熟悉的数据处理语言去访问数据库。例如，把异构数据库系统中所有的非关系型局部数据库均转换为关系数据库系统，因此，非关系数据库用户就可以用关系数据处理语言访问所有的局部数据库系统。对于关系数据库系统用户而言，这种访问和访问一个关系数据库一样，称之为数据库访问的透明性。

为了达到访问的透明性，数据库转换所追求的是把多个不同的局部数据库转换为同一数据模型的数据库。有关研究表明，把一个语义丰富的数据模型转换为语义缺乏的数据模型是可行的，同样把一个语义丰富的数据模型转换为语义更为丰富的等价的数据模型也是可行的。例如，面向对象数据模型转换为关系模型；层次模型转换为关系模型；层次模型、网络模型、函数模型等向基于属性的数据模型的转换是可行的。反之，语义欠缺的模型向语义丰富的模型的转换也是显然可行的。然而，如何实现数据模型间的转换则是数据库转换的关键。

实现数据的透明访问可以采用多对一转换、双向的中间件等技术。ODBC 是一种用来在相关或不相关的数据库管理系统中存取数据的标准 API，允许用户书写可以运行于来自不同厂商的数据库服务器上的应用，为应用程序提供了一套与产品无关的高层调用接口规范和基于动态链接库的运行支持环境。目前，常用的数据库应用开发的前端工具（如 PowerBuilder、Delphi 等）都通过 ODBC 接口来连接各种数据库系统。而多数 DBMS（如 Oracle、Sybase、Microsoft SQL Server 等）都提供了相应的 ODBC 驱动程序，使数据库系统具有很好的开放性。

ODBC 接口的最大优点是其互操作能力。理想情况下，每个驱动程序和数据源应支持完全相同的 ODBC 函数调用和 SQL 语句，使得 ODBC 应用程序可以操作所有的数据库系统。然而，实际上不同的数据库对 SQL 语法的支持程度各不相同，因此，ODBC 规范定义了驱动程序的一致性级别，ODBC API 的一致性确定了应用程序所能调用的 ODBC 函数种类。例如，ODBC 2.0 就规定了三个级别的函数。

随着 Internet 应用的不断普及，Internet 的异构分布式信息系统正在迅速发展。Java 以其平台无关性、移植性强、安全性高、稳定性好、分布式、面向对象等优点而成为 Internet 应用开发的首选语言。在 Internet 环境下，实现基于异种系统平台的数据库应用，必须提供一个独立于特定数据库管理系统的统一编程界面和一个基于 SQL 的通用的数据库访问方法。Java 的数据库接口规范 JDBC 是支持基本 SQL 功能的一个通用的 API，它在不同的数据库功能模块的层次上提供了一个统一的用户界面，为对异构数据库进行直接

的 Web 访问提供了新的解决方案。JDBC 已被越来越多的数据库厂商、连接厂商、Internet 服务厂商及应用程序编制者所支持。

5.2 异构数据库互连

当前，数据库的应用范围随着计算机科学技术的发展与普及不断扩大，数据库技术也在飞速发展，各种新产品、新技术不断出现，尤其以关系型数据库为代表的数据库产品已逐渐走向成熟。人们对信息的需求也越来越广泛，而且这种需求已不再局限于一个部门内数据库的相互访问，有时还涉及到部门之间甚至行业之间的数据共享。因此，当今用户所面对的是一个多厂商异种数据库、异种操作系统和异种网络环境的态势，异种数据库间互连已经成为人们越来越迫切的需求。

然而，异种数据库的互连并不是一件容易的事。

5.2.1 数据库之间的差异

对于实际运行的系统来说，有许多因素都可能产生数据库系统之间的差异性，如计算机硬件、操作系统、网络通信协议、DBMS 及数据模型等。其中，源自数据库系统自身的差异可以分为两大类，即 DBMS 的差异和数据语义的差异。

（1）DBMS 的差异。数据模型的不同是 DBMS 差异的一个重要方面。关系数据库、层次数据库、网状数据库以及新出现的面向对象数据库，它们所采用的数据模型各不相同，由此而导致数据结构、约束和数据语言等的差异。

（2）数据语义的差异 。数据语义的差异主要源于不同数据库对相同或相关数据的理解、解释及使用的不一致性。例如，在两个数据库中，对同一个属性名的具体含义、定义不同或对同一属性的数据值在两个数据库中的精度定义不同，都可能引起语义差异。

由于目前关系型数据库管理系统（Relational DataBase Management System，RDBMS）是数据库产品的主流，所以，下面主要针对关系型数据库之间的互连进行讨论。

各数据库厂商的 RDBMS 产品名义上都符合 ISO 规范，但是由于 SQL 标准并不完善，加之各厂商为了提高自己产品的功能都对 SQL 语言进行了一定的扩充。例如，Oracle 的 DL/SQL、Sybase 的 T-SQL 和 Informix 的 I-SQL 等，就连 IBM 公司的 4 个基于 SQL 的 RDBMS-DB2、SQL/DS、SQL/400 和 OS/2 Database Manager 彼此也不兼容，这就给数据库之间的互连造成了很大的障碍。

为了解决异种数据库之间的互连问题，国际标准化组织和各数据库厂家已经作了不懈的努力。SQL 访问集团（SQL Access Group，SAG）试图从标准一致化的角度来解决这个问题，并制定了一系列规范。例如，正在广泛使用的 ODBC 即是基于 SAG 的调用级接口（Call Level Interface，CLI）规范。数据库厂家为使自己的产品具有开放性，也提供了各自的 Gateway（网关）产品，用以提供对异种数据库的透明访问。

5.2.2　SAG 与 DRDA

SAG 和 IBM 公司为解决异种数据库互连，各自建立了实现基于 SQL 的异种数据库互连的规范。

1．SAG

SAG 成立于 1989 年，它的目标是建立基于 SQL 的异种关系型数据库互操作的技术规范。SAG 由 40 多家公司组成，其中包括 DEC、HP、Sun、Sybase、Informix、Borland 和 Microsoft 等，其规范已被 X/Open 所采纳。SAG 和它的许多成员与美国国家标准局（American National Standards Institute，ANSI）和 ISO 合作，对正式的 SQL 标准进行了改进。1991 年 12 月，TCP/IP 加入其规范中，在此之前，SAG 规范已包括了符合开放系统互联（Open System Interconnection，ISO/OSI）模型的网络。

SAG 已产生的两个规范如下。

（1）API 规范：定义了一个嵌埋式数据库语言规范。该规范基于 ANSI 和 OSI 的 SQL 定义（ANSI X3.135-1989 和 X3.168-1989）。

（2）格式与协议（Format And Protocol，FAP）规范：用于数据库应用与 RDBMS 间的通信。该规范利用了 ISO 远程数据库访问（Remote Data Access，RDA）委员会的工作。

SAG 规范对其作为基础的标准进行了丰富和完善，对原标准中认为应由实现者定义的部分进行了明确的定义。例如，它们较严格地限制了实现者的多种选择，使按此写出的可移植的应用程序在上述限制工作的系统上可以互相操作。

2．DRDA

分布式关系数据库体系结构（Distributed Relational Database Architecture，DRDA）是在 IBM 和非 IBM 平台上访问数据库信息的 IBM 标准。DRDA 遵循 SQL 标准，它是 IBM 的信息仓库框架的关键部件，为 IBM 的 RDBMS 定义了应用编程界面、数据格式和网络协议。DRDA 参与了将基于局域网的数据库系统与基于 IBM 大型计算机的数据库系统，如 DB2、SQL/DS 和 SQL/400 数据库系统互连。已有包括 HP、Oracle 等多家厂商声称，他们的产品将支持 DRDA，他们采用的方式是做 DRDA 的客户。对于实现分布和连接多个 DB2 的大型数据库来说，DRDA 是很可取的，但也存在一些缺陷，其中最明显的是只对 IBM 透明，而且 DRDA 是单向的，不能借助它来访问非 IBM 数据。

SAG 规范和 IBM 的 DRDA 都是为了解决基于 SQL 的 RDBMS 互操作问题而进行的努力，它们走的是两条不同的道路。从技术上看，它们的区别如下。

（1）语言：SAG 规范和 DRDA 都使用 SQL 作为查询语言，但使用的方法不同。SAG 采取了一个公共集的途径，被使用的是一个单一的语言定义。这种一致的 SQL 语法和语义，使希望得到可移植应用的开发者们不必在各种语言中寻找一个公共子集，同时，使目标服务器的选择可延缓到运行时才进行；在 DRDA 环境中，访问三个不同服务器的同

一个客户应用，可能要在同一程序中包含三种不同的 SQL 变种。

（2）消息/数据值的编码：SAG 规范使用 OSI 标准；而 DRDA 使用 IBM 体系结构。

（3）目录表：SAG 规范定义了一个具有标准属性和值的目录表集合，这些表基于相应的 SQL2 定义；DRDA 对目录管理表采取的是“哪一个都行”的策略。

（4）网络需求：SAG 规范与 DRDA 有各自的通信网络环境，SAG 规范基于 OSI 参考模型，并采用 OSI 的寻址和命名结构。

（5）数据转换：在 DRDA 中，由消息接收者来执行发送者的数据格式与其自身平台上的格式之间的转换；在 SAG 规范中，被传送的值是以一种定义良好并独立于平台的标准形式来表示的，发送方把自己的数据转换成标准格式，而接收方把标准格式转换成自己平台上使用的格式。

另外，SAG 对标准的改进建议，由代表提交给 ANSI X3H2（SQL）和 X3H2.1（RDA）委员会进行表决。而 DRDA 是 IBM 的一个体系结构，如果有足够多的公司支持它，那么一段时间之后，就有可能成为事实上的标准。SAG 规范由国际标准化组织与厂商组成的联盟控制，DRDA 则归 IBM 所有，它的规范无疑是受 IBM 控制。DRDA 的改进程序由 IBM 的一个内部体系结构委员会管理，其开放性远不如 SAG。

由此可见，SAG 的规范会沿着更为开放的道路发展，而 DRDA 则要封闭得多。事实上，基于 SAG 规范的 ODBC 的出现，在一定程度上实现了异种数据库之间的互联。

5.2.3 ODBC 与 JDBC

ODBC 是 Microsoft 公司提出的一个关于开放数据库互连的体系结构，这是一个基于 SAG 的 CLI 草案而提出的数据访问标准，其目的是使 Windows 具有对位于任何地方的其他各种异构数据库系统的访问能力。

虽然 ODBC 是独家制定的技术规范，但由于 Microsoft 公司本身就是 SAG 的成员之一，它用 ODBC 来表示对 SAG CLI 的支持，同时得到了 SAG 其他成员的支持。目前，ODBC 已被数据库界广泛接受，成为事实上的工业标准。ODBC 允许与 IBM 公司的专用技术规范 DRDA 连接，因而增加了其开放性。因此，Oracle、Sybase、Informix 及 SQL Server、Foxpro 等都提供了对 ODBC 的支持。对用户而言，由于 ODBC 提供的是一种独立于数据库的应用编程接口，解决了嵌入式 SQL 接口（SQL API）非规范化的矛盾，提供了 SQL API 规范的核心，免除了应用软件随数据库的改变而改变的痛苦，所以若要访问新的数据源，只要安装与其相对应的驱动程序即可，从而可以大大节省用户投资。

ODBC 的基本思路是为用户提供简单、标准和透明的数据库连接的公共编程接口，而由开发商根据 ODBC 的标准去实现底层的驱动程序，这个驱动对用户是透明的。

1．ODBC 的体系结构

ODBC 的体系结构共分以下 4 层。

（1）应用程序（Application）：负责调用 ODBC 函数来提交 SQL 语句，提取结果。

（2）驱动程序管理器（Driver Manager）：为应用程序加载驱动程序。

（3）驱动程序（Driver）：处理 ODBC 函数调用，向数据源提交 SQL 请求，向应用程序返回结果，必要时驱动程序将 SQL 语法翻译成符合 DBMS 语法规定的格式。

（4）数据源（Data Source）：由用户想要存取的数据、操作系统、DBMS、网络平台等组成。

对应用程序来说，在处理函数调用时，驱动程序和驱动程序管理器是一个整体。

如果一个驱动程序声明它支持某一个符合性级别，那么该驱动程序就应该支持该符合性级别中定义的全部功能，而不管这些功能是否被相应的 DBMS 支持。符合性级别并不限制驱动程序只能支持规定的功能，而是鼓励驱动程序开发者支持尽可能多的功能。

ODBC 将函数调用划分为三级，大多数应用程序都要求驱动程序至少支持一级 API，为了保证 ODBC 驱动程序在应用程序中较好地完成工作，驱动程序开发者应实现一级 API 中的所有功能。

（1）核心 API：它包括了与 SAG 的 CLI 相匹配的基本功能；分配与释放环境句柄、连接句柄及语句句柄；连接到数据源；准备并执行 SQL 语句或立即执行 SQL 语句；为 SQL 语句的参数和结果分配缓冲区；提取结果及有关信息；提交或撤销事务处理；提取错误信息。

（2）一级 API：它包括了核心 API 的全部功能；用驱动程序规定的对话框连接数据源；传送一个参数值的部分或全部；提取一个列值的部分或全部；提取词典信息（列、统计、表等）；提取驱动程序和数据源的兼容性信息。

（3）二级 API：其功能包括核心级和一级 API 的全部功能；浏览连接信息和可用的数据源清单；传送参数值数组，提取结果数组；提取参数个数及单个参数的描述信息；使用可滚动光标；提取 SQL 格式；提取词典信息（权限、关键词、过程）；调用翻译程序 DLL。

SQL 语法的符合性级别也有三个，它们分别如下。

（1）最小级：数据类型只能使用 Char 类型。数据定义语言 DDL 可以使用创建数据表（Create Table）、删除数据表（Drop Table），但不能修改表结构，也不能对表建立索引或视图；数据操纵语言 DML 能支持一般的 Select、Insert、Delete 和 Update 子句，也支持简单的表达式。

（2）核心级：能使用全部标准的数据类型。DDL 可以创建表、删除表、修改表结构、建立或删除索引及视图，并具有 Grant 和 Revoke 等安全性功能；DML 比最小级功能增加了连接定位更新、定位删除和子查询等，并支持聚集函数，如 SUM。

（3）扩展级：除了包含核心级提供的功能外，还能使用二进制数据类型。能进行外连接查询，支持批处理 SQL 语句进行过程调用等。

2．ODBC 接口

ODBC 接口定义了如下内容：

（1）ODBC 函数库。利用这些函数，应用程序可以连接 DBMS，执行 SQL 语句，提取查询结果。

（2）SQL 语法。遵循标准“X/Open and SQL Access Group（SAG）SQL CAE specification（1992）”。

（3）错误代码。

（4）连接、登录 DBMS。

（5）数据类型。

ODBC 接口具有相当的灵活性，构成 SQL 语句的字符串可以在源程序中直接给出，也可以在运行时动态生成；同一个程序可以存取不同的 DBMS；应用程序不必关心 ODBC 与 DBMS 之间的底层通信协议；数据值可以用应用程序最方便的格式传递。

ODBC 接口提供两种类型的函数调用：

（1）核心函数。基于标准“X/Open and SQL Access Group Call Level Interface specification”。

（2）扩展函数。支持附加的函数，包括光标控制和异步进程。

ODBC 方式同传统方式相比，不管底层网络环境如何，不论采用何种 DBMS，用户在程序中都使用同一套标准代码，无需逐个了解各 DBMS 及其 API 的特点，源程序不因底层的变化而被修改，从而减少了开发维护的工作量，缩短了开发周期。

3．ODBC 的缺点

目前，在许多产品中，如 PowerBuilder、Visual Basic、Visual C++、Excel 和 MS Access 等，用户只要安装不同的 ODBC 驱动程序，就可存取相应的数据库产品，而不管采用何种前端开发软件，也不管后台是何种数据库，其存取过程是一致的。但 ODBC 也存在着一些问题。例如，由于规定了三个层次的一致性级别，应用程序与驱动程序之间的匹配会出现一些问题和矛盾；由于 ODBC 的层次较多，性能比专用的 API 差；由于 ODBC 为不同的开发商开发，测试工作不能统一等。但是，ODBC 作为一项很重要的技术，与 ODBC 厂商和第三方厂商已建立了密切的合作，并将进一步完善该技术。因此，ODBC 作为开发数据库连接的工业标准，在客户/服务器结构的网络环境中得到了广泛的应用。

4．各厂家的数据访问接口

Microsoft 数据访问构件（Microsoft Data Access Components，MDAC）由 ActiveX 数据对象（ActiveX Data Object，ADO）、远程数据服务（Remote Data Service，RDS）、Microsoft OLE DB Provider for ODBC、ODBC、Microsoft SQL Server ODBC 驱动程序、Microsoft Access 以及其他数据库和 Oracle 数据库组成。对象链接和嵌入数据库 OLE DB（Object Linking and Embedding DataBase）是在企业和 Internet 上提供对 SQL 和非 SQL 数据源访问的数据访问接口。

除了 Microsoft 公司的 ODBC 外，其他厂商也开发了一些数据访问接口。

集成数据库应用程序编程接口（Integral Database API，IDAPI）起初是由 Borland 公

司开发的，后来其他厂商对它提供了支持，IDAPI 功能与 ODBC 类似，并且也是围绕着调用层接口设计的。Borland 认为，IDAPI 支持的服务器较 ODBC 多，但是现在 ODBC 拥有更多的工业支持。

数据访问语言（Data Access Language，DAL）是 Apple 公司开发的数据访问语言，用于 Macintosh 用户访问各种后台数据库产品，包括 IBM 大型计算机和中型机数据库。DAL 与 SQL 相关，DAL 客户机软件可在 Windows 下运行，并允许 ODBC 应用程序访问所有 DAL 能够访问的数据库和平台。

Oracle 的 Glue 是一种通信 API，包括一系列的访问后台数据库服务器和通过平台连接信息系统的命令。Glue 访问数据较 ODBC 简单，它是为多个前台应用程序访问 Oracle 数据库而设计的。

JDBC 是与 Java 语言一起使用的基于 ODBC 的数据访问接口。Java 提供了不同的程序接口用于连接数据库和执行 SQL 语句。利用 JDBC 接口，可以执行通常的 SQL 语句、动态 SQL 语句以及带 IN 和 OUT 参数的存储过程等。

5.2.4　利用网关互连

多厂商异种数据库及系统的存在为用户提供了众多的选择机会，但所引起的互操作问题又影响了用户的自由选择。数据库网关可以帮助解决这一问题。

1．网关的概念

数据库中的网关借用了计算机网络中网关的概念。数据库网关允许一个本地 DBMS 用户访问另一个相同或不同平台上的 DBMS，用户不必知道数据库所使用的存取机制。所以数据库网关实际上相当于界面转换器，其逻辑成分包含以下两个部分。

（1）客户 API 库：客户利用它向服务器提交远程数据请求，并处理服务器的响应。

（2）服务器 API 库：是客户 API 库的镜像。客户 API 的子例程发出请求，而服务器 API 库的子例程则产生对应于这些请求的事件，同时利用它返回结果。

2．Sybase 网关

在分布式数据库系统中，异构的或统一的系统，其开放性和透明性都十分重要。Sybase 通过客户端/服务器结构框架，为解决分布式数据库系统的开放及透明问题提供了一系列方法。Sybase 的分布式数据库可互操作性解决方法的发展，经历了三个阶段。

第一个阶段，开放界面。这是解决异构可互操作性最开放、最灵活的途径。Sybase 的 Open Client 和 Open Server 同时提供客户编程界面，允许用户将许多应用和工具与任何可用的数据资源建立联系。具有这种功能的开放界面将为程序员和前端用户在硬件、操作系统、网络和数据库系统方面提供高度的透明性，并能提供高层可靠性和可达性，以及局域自治性和连续操作等性能。显然，这里并未涉及位置透明性问题，因而，用户和程序员必须知道数据的存储位置和结构，各成员库之间的关系必须在应用中编码，任何分布式事务的约束也必须写入应用代码。

第二个阶段，SQL Passthru 网关。这种网关把 SQL 语句传递到各成员库的 RDBMS 中，在那里执行 SQL 并检索数据。这类网关必须提供两层服务：上层对开发者提供的服务比开放界面提供的服务简单，一般只包括 SQL 编程界面；下层的运行服务同样是解决异构硬件、操作系统、网络和数据库系统的差别问题，但增加了对特定的源和目标 RDBMS 数据及信息的自动翻译。该网关允许用户通过使用他们熟悉的目标 RDBMS 工具和技术来访问外界的数据源。其缺点仍然是没有解决位置透明性和 SQL 透明的问题。

第三个阶段，Sybase OmniSQL 网关。这种网关是一个 Open Server 应用程序，它提供到各异构分布数据源的透明的 T-SQL 界面。Sybase OmniSQL 网关最初曾用 Open Server 2.0 开发。从效果上讲，Sybase OmniSQL 网关提供了数据位置和分布 DBMS 要求的 DBMS 透明性。外部数据源包括在各数据库系统宿主数据文件和表格之中。OmniSQL 网关也可被用于访问 Ingress 数据库和被存储在 Novell 的 Btrieve 记录管理程序中的数据。Sybase OmniSQL 网关提供对所有受支持的异构数据源的透明的读/写访问。

OmniSQL 网关是可以同时支持对多个异种数据源进行读写访问的通用网关，具有单一的开发管理环境和数据访问的语言，同时也可作为一个 Passthru 网关使用。它提供了非常出色的位置透明性、SQL 透明性，并将强大的 Sybase T-SQL 特征扩展到 Oracle、DB2、IMS 和 ISAM 等数据源，且能够支持功能强大的全局存储过程。

3．Oracle 网关

目前，Oracle 利用透明网关实现了和 MS SQL Server、Sybase、DB2 等多种数据库的互连。在 Oracle 和 SQL Server 之间使用 Oracle 透明网关服务器实现互连互通，其中透明网关服务器可以与 Oracle 或 SQL Server 数据库在同一台主机上，也可以是在独立的一台主机上。

Oracle Open Gateway 技术提供了 Oracle 数据与非 Oracle 数据的联合操作，并可服务于 Oracle 逻辑数据库。用 SQL*Net 可以连接客户和服务器或服务器和服务器。在异构环境中，SQL*Net 还起着服务器与网关连接的作用。

用 Oracle 开放的网关技术可将异构的数据库系统连接在一起，构成一个协调的服务器结构，从而将各种不同的数据库作为一个逻辑数据库提供给用户。Oracle 的开放网关技术提供透明的及过程的两种类型的网关。两者的区别在于，一种是 Oracle 提供的，另一种是用开发者工具实现与非 Oracle 数据源的接口。Oracle 提供的 SQL 网关是 Oracle 产品的透明网关族，如 SQL*Connect，而用户可以完成的 SQL 网关是 Oracle 的透明网关开发者工具（Transparent Gateway Developer's Kit）。Oracle 提供的过程网关是为指定的过程系统所做，如事务处理系统等。

（1）透明网关：透明网关通过 SQL 语言存取数据。这些网关以所存取的非 Oracle 数据的范畴而分类，其中有 SQL 或基于集合的数据库管理系统、基于层次的系统和以记录存取的文件系统。

（2）过程网桥：过程网桥提供了 PL/SQL 远程调用功能，允许用户用第三代语言实

现过程及函数或在事务系统中访问服务器。过程网桥提供如下功能：PL/SQL 数据类型及主语言数据类型的转换；基于规则的例外报告及参数的平滑过滤；用户所写的子程序对每个过程或函数的调用；事务协调。

4．Informix 网关

Informix 网关支持 DRDA 标准。利用此网关，用户和开发者可以透明地访问 DB2、DB2/VM 和 DB2/400，就如同访问一个 Informix 数据库服务器一样。

支持 DRDA 标准的 Informix 网关提供了第一个基于 UNIX 的对 IBM 关系型数据库进行互联的解决方案，采用此方法，不需要购买附加的主机软件就可灵活地读写。对于终端用户而言，Informix 网关是完全透明的。它就像一个翻译器，可以访问基于 UNIX 或 Windows 平台应用中的 IBM 数据。从概念上讲，该网关对应用而言就像一个 Informix 数据库服务器，而对于 IBM 数据库而言，它就像一个 DRDA 客户应用。因为支持 DRDA 标准的 Informix 网关对于用户来说是一个 Informix 数据库服务器，因此，能够访问 Informix 的第三方工具就可以访问 IBM 的 DB2、DB2/VM 或 DB2/400 SQL 数据库。

Informix 网关在 Informix 客户和 IBM 数据库服务器之间充当翻译器，管理两者之间的网络连接。当从客户端发出 SQL 请求时，执行以下步骤：

（1）网关将 Informix 应用的 SQL 请求翻译为 DRDA 格式和协议。在将不同的数据类型转换成相应的 DRDA 格式之后，原来的 SQL 请求直接传送给 IBM DRDA RDBMS。

（2）IBM RDBMS 处理请求，将执行结果或状态码返回给网关。

（3）网关将 DRDA 结果翻译为适于 Informix 客户应用的格式和协议，执行必须的数据转换，然后将结果或状态码返回给应用程序。

Informix 网关具有分布处理能力，它使 Informix 数据库和 IBM 数据库得以集成。当访问多个不同的数据库时，开发者可以使用强大的 Informix 工具。

Informix 还有其他对 Oracle、Sybase 等的网关，原理基本相同，不再详述。

5.2.5　数据库互连方法发展展望

采用数据库网关可以比较满意地解决数据库的互连问题。除此之外，数据库网关还有不少有利之处。例如，通过网关可以把第三方为其他厂商开发的工具连接到自己的数据库产品上；用户无需废弃现有的应用程序，利用网关可把它们与一些新的数据库技术（例如，面向对象的 DBMS 等）集成起来，从而保护用户过去的投资。但是，n 个异种数据库要实现任意两个数据库间的互联，就必须提供 $n \times (n-1)/2$ 个网关，这在实际应用中是很难实现的。而且有些异构数据库间的数据格式、语法或语义的转换也是行不通的，利用数据库网关，动用远地异构数据库不易达到完全透明。因此有人说，数据库网关仅是连接异构数据库、实现互操作的权宜之计。一旦统一的应用界面标准被广泛采纳，如 5.2.2 节讨论的 SAG 和 DRDA 等，就不再需要网关了。

但是，即使“完善”的标准出现，符合这些标准的数据库产品也未必就能实现互连

操作。由于商业竞争的需要，各厂商为保持自己的地位，必然在自己的产品中加入独有的特征，从而产生新的不兼容性，为互连造成新的障碍。从上述情况来看，作为解决数据库互连操作的一种实际方法，数据库网关还是可行的。那么，异种数据库的互连是否能完全解决呢？人们发现，正是由于关系数据库系统内在的不可克服的缺陷，导致了互连操作的难以实现。现在面向对象的数据库技术、对象-关系数据库技术、数据仓库技术等为异构数据库互联提供了新的思路。

数据仓库可以是异构数据库系统中的多个数据库，并建立统一的全局模式，同时支持对历史数据的访问，用户通过数据仓库提供的统一的数据接口进行决策支持的查询。在数据仓库的基础上，还可以进行数据挖掘、Web 挖掘，实现真正的信息检索查询。

目前，异构数据库系统的集成以及建立在此基础之上的数据仓库、数据挖掘已经成为网络数据库技术研究的重点之一。著名的国内外数据库厂商也将异构数据库系统作为竞争的焦点，研究如何将原来传统的、可能分布于各地的多个关系数据库集成起来进行改进和发展，形成虚拟异构数据库系统和数据仓库，更好地为企业信息化、电子商务服务。

本章参考文献

[1] 周英飚，许蔚. 异构数据库体系结构. http://tech.csai.cn/dbms/NO000033.htm

[2] 吴勉，张顺颐. 异构数据库互联. http://tech.csai.cn/dbms/NO000032.htm

[3] 文必龙，邵庆. 开放数据库互联（ODBC）技术与应用. 北京：科学出版社龙门书局，1997

[4] 李昭原. 数据库技术新进展. 北京：清华大学出版社，1997

第 6 章　商业智能与数据仓库

信息技术应用的卓越成效在经过 20 多年的信息化建设进程中已经初步显现，信息技术广泛应用的同时带来了信息的泛滥，正如 John Naisbett 所说，“我们已被信息所淹没，但是却正在忍受缺乏知识的煎熬”。如何从大量的数据中快速有效的提取出用户需要的信息和知识也显得越来越重要，让用户不至于被信息海洋所淹没；如何有效利用多年信息系统积累下来的大量数据，从被深埋的历史数据中挖出财富；如何解决信息化建设过程中，由于历史的认识水平和技术条件的限制所造成的信息化各子系统的脱节，而直接导致的信息孤岛问题。以上这些都是当今政府和企业的信息系统迫切需要解决的问题。

“工欲善其事，必先利其器”，商业智能正是为了解决上述问题而应运而生的，商业智能本身是一个庞大的技术体系，也是一个还在发展中的概念，如何理解商业智能的内涵，业界有不同的观点，而本章从工程实践的角度，把商业智能看成实现“数据→信息→知识→行动→智慧”过程所用到的技术和方法，所以本章内容在组织安排上，体现了两种对商业智能的观点。

（1）一种观点把商业智能看成是一个过程，这是 DWReview 的观点，商业智能是帮助企业实现“数据→信息→知识”的过程，这是本章 6.1.1 节至 6.1.3 节所描述的内容，力求读者对商业智能有一个技术视野以外的认识。

（2）另外一种观点是把商业智能看成一系列的技术和方法，这是代表了最早由 Gartner Group 于 1996 年提出商业智能定义的观点：商业智能为一类由数据仓库（或数据集市）、查询报表、数据分析、数据挖掘、数据备份和恢复等部分组成的、以帮助企业决策为目的技术及其应用，这是本章 6.1.4 节和其他几节所要描述的内容，具体描述各个相关技术点的知识。

6.1　商业智能概述

商业智能既不是空穴来风，也不是无中生有，而是来自于应用的需求。本节将从时代发展需要和现实需求的角度，对应用商业智能的深层原因进行探讨。

6.1.1　商业智能的来龙去脉

企业与政府机构的信息化建设从零开始到现在，大致经历了三个阶段，如图 6-1 所示。

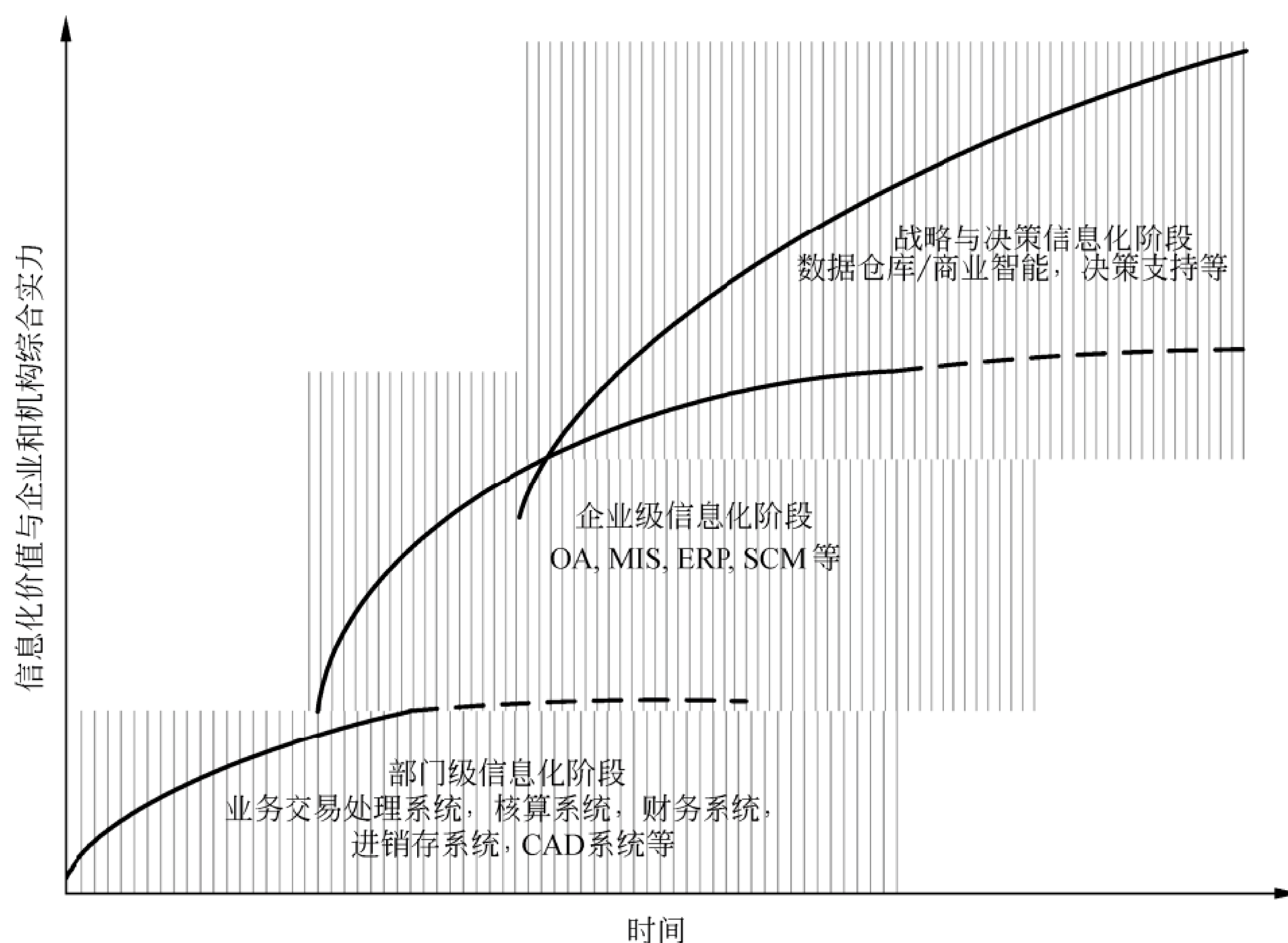


图 6-1　企业信息化建设的三个阶段

第一阶段是部门级信息化阶段，在这一阶段，机构内往往是一些需求最迫切的部门首先采用了信息技术，如财务系统、计算机辅助设计系统、电子数据交换系统等等，以电子化和处理自动化来取代低效繁杂的手工处理；第二阶段是企业级信息化阶段，这一阶段政府机构与企业往往已经拥有了几个分别建设的业务处理系统，机构期望从总体角度建设高度集中的、或互相联接的综合业务管理系统，如管理信息系统，企业资源计划、办公自动化等；第三阶段是企业或政府机构的战略与决策信息化阶段，这时企业或政府机构往往已经建成了比较完整的业务处理系统，而企业和政府机构在对业务系统里面数据的综合利用主要面临着三大问题，分别是不同业务数据的共享与综合处理问题、历史数据的利用问题以及数据角度的事务处理与决策差异性问题，这三大问题也是满足企业或机构的领导者和决策者通过数据来做出正确判断和决策的障碍，因此建立专门面向各级领导与决策层的信息系统，实现企业或政府机构战略和决策的信息化，可以说是这一阶段的核心任务。

可见，随着企业的发展所带来的对信息需求分别从广度和深度两个层面不断扩展，第一阶段的发展瓶颈在于信息的空间局限性，第二阶段的发展瓶颈在于信息的时间局限性，可以说，企业信息化进程经过这三个发展阶段，企业的管理水平也从“基于数据”，发展为“基于信息”，进而上升为“基于知识”。显然，在竞争日益激烈的环境下，错误

的决策对企业的打击是颠覆性的，一个企业只有达到“基于知识”的学习型组织的管理境界，才能不断成长，在市场上生存。

把企业形象地比喻为一个人的话，第一阶段是针对手指的自动化，第二阶段是针对眼睛和耳朵的自动化，第三阶段是针对大脑的自动化。从图 6-1 的纵坐标也可以看到，信息化的价值以及由此给企业带来的实力也是获得提升的。而第三阶段正是商业智能走上历史舞台并且成为主角的时代，也是企业求生存求发展不得不走向学习型组织管理方式的历史选择，促成商业智能产生和发展的根源，笔者认为，不单纯是技术的进步，当然不可否认，技术的进步创造了物质上的条件，然而从因果关系的角度来说，企业选择商业智能的根本原因，是市场竞争下求生存谋发展的必然。

6.1.2　什么是商业智能

商业智能的概念最早是 Gartner Group 于 1996 年提出来的。其实，商业智能所涉及的技术与应用，在 Gartner Group 命名之前就有，起初被称为领导信息系统（Executive Information System，EIS），在羽化成商业智能之前也称为决策支持系统（Decision Support System，DSS）。

从技术层面上讲，商业智能或数据仓库（Data Warehousing，DW）并不是什么新技术，它只是数据库技术、联机分析处理（On-Line Analysis Processing，OLAP）技术、数据采集和迁移技术、网络技术、GUI 技术、查询与报表技术、统计学、人工智能、知识发现技术等理论和技术的综合运用，从这个意义上，把商业智能看成是一种体系结构应该比较恰当。关于体系结构与具体技术的关系，W.H.Inmon 形象地比喻成新墨西哥州的一个城市圣达菲和砖块的关系，圣达菲这个体系结构由砖块和裸露的横梁构成，没有这些砖块就没有圣达菲的各种建筑，而砖块本身并不能构成圣达菲这个体系。

商业智能的核心内容是从许多来自企业不同的业务处理系统的数据中，提取出有用的数据，进行清理以保证数据的正确性，然后经过抽取（extraction）、转换（transformation）和装载（load），即 ETL 过程，整合到一个企业级的中心数据仓库里，从而得到企业信息的一个全局视图，在此基础上利用合适的查询和分析工具、数据挖掘工具等对数据仓库里的数据进行分析和处理，形成信息，更进一步把规律性的信息提炼成知识，并且把对决策有帮助的信息和知识呈现给管理者，为管理者的决策提供支持。商业智能的这个基本过程如图 6-2 所示。

数据仓库是商业智能的基础，商业智能的应用必须基于数据仓库技术，所以数据仓库的设计工作占一个商业智能项目的核心位置，在很多项目命名时，往往是把数据仓库和商业智能相提并论，要么把它们等同起来，有时这会给人一种很混淆的感觉，造成了很多初学者在认识上的误区。一般来说，上面所描述的是一个广义上的商业智能概念，在这个概念层面上，数据仓库是其中非常重要的组成部分，数据仓库从概念上更多地侧重在对企业各类信息的整合和存储工作，包括了数据的迁移、数据的组织和存储、数据

的管理与维护等，这些平常称之为后台的基础性的数据准备工作；与之对应，侠义的商业智能概念则侧重在数据查询和报告、多维/联机数据分析、数据挖掘和数据可视化工具，这些平常称之为前台的数据分析应用方面，其中数据挖掘是商业智能中比较高层次的一种应用。图 6-3 表达了商业智能过程中对应使用的技术和方法。

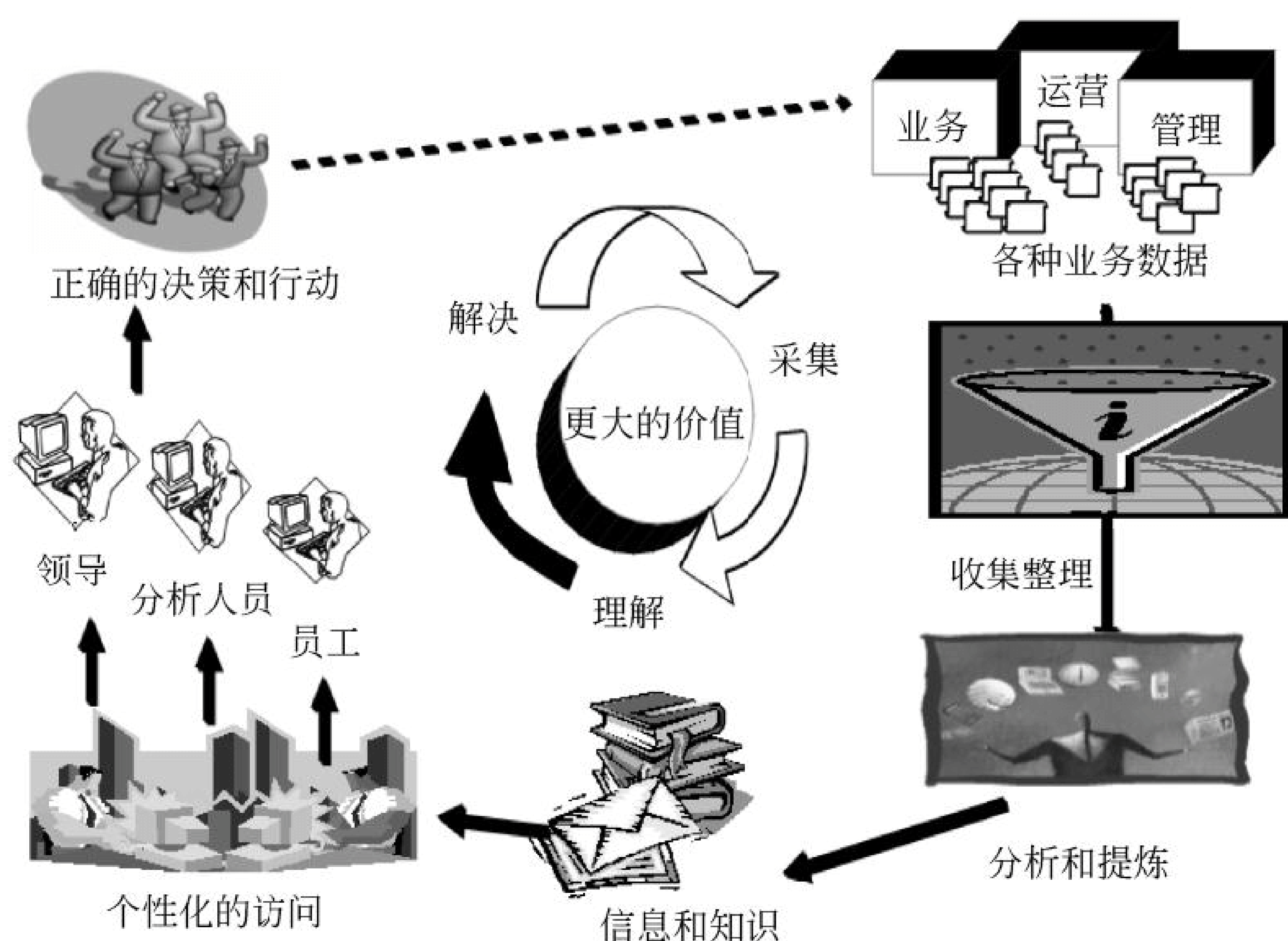


图 6-2　商业智能的基本过程

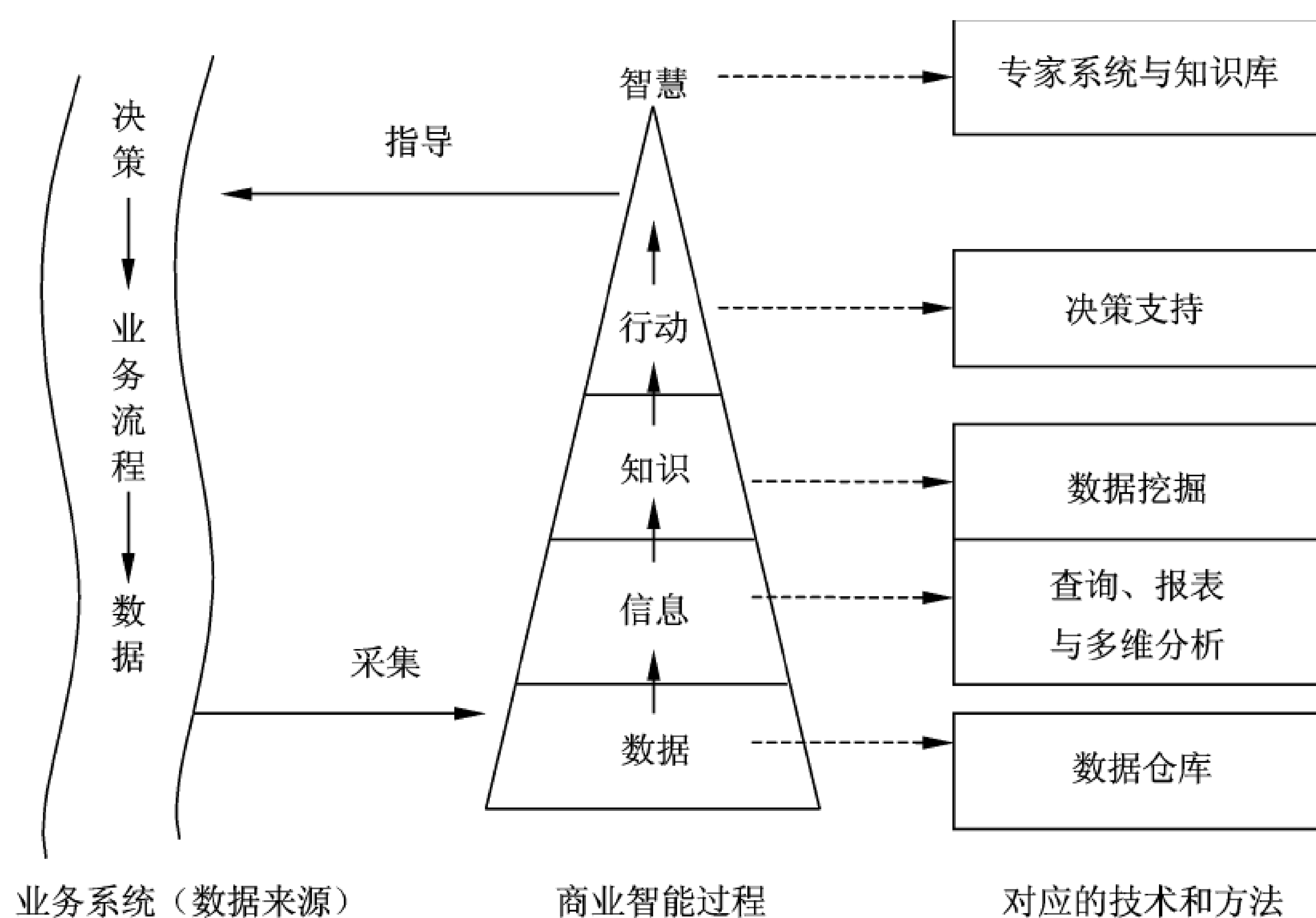


图 6-3　商业智能过程及其对应的技术和方法

6.1.3　商业智能的需求

商业智能系统的服务对象包括企业或组织机构的决策人员、数据分析专家、中下级别经理和一般业务人员，而不同层次的用户对商业智能应用的需求有明显的差异。

高层决策者需要了解业务的总体情况和总的发展态势，他们可能使用系统提供的分析工具自己发现问题，但更主要的是利用分析结果进行决策，高层决策者需要通晓业务的具体状态和发展趋势，包括业务的状态和构成（机构构成、时间构成、产品构成、客户构成等等）以及对业务的发展趋势做出预测。

数据分析专家需要更加深入地从数据仓库的数据中发现问题、市场机会和风险，需要及时把发现的结果报告给高层决策者。

中下级经理和业务人员通常仅仅关心与各自工作相关的内容，他们或许对报表和固定的数据查询更为习惯。

图 6-4 描述了商业智能系统中各种用户角色对系统数据深度、广度、分析复杂性、对目标软件易用性，对软件的控制能力和客户化程度要求以及对业务整体和局部信息需求程度的要求。

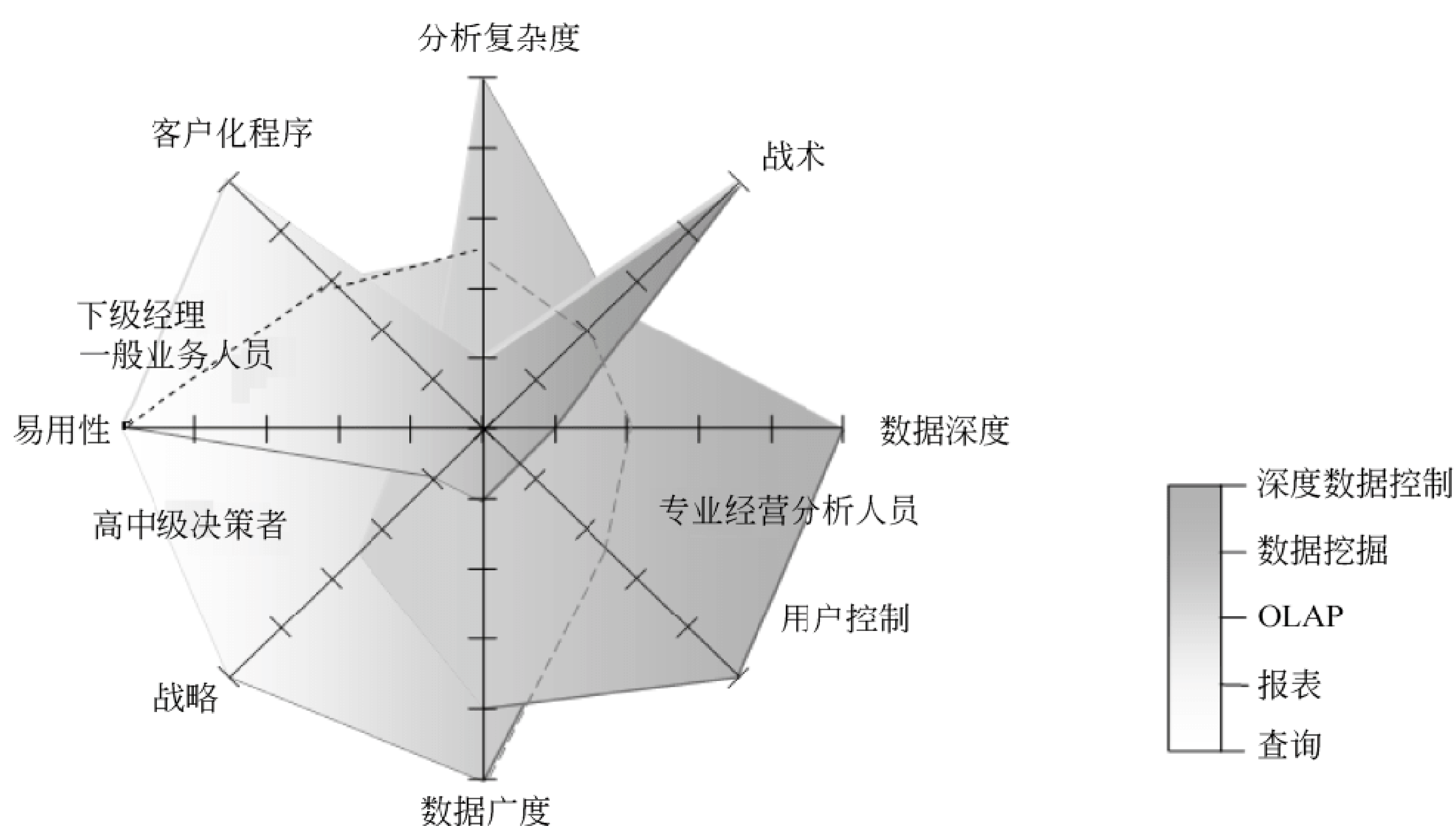


图 6-4　商业智能用户类型分析图

分析用户类型是系统功能设定、分布的依据。图中以色谱形式表示对信息服务深度的需求，从最浅显的数据查询到深度数据挖掘。8 条坐标线表示用户对不同系统特性的需求。这些系统性能是：数据深度和广度、分析的复杂性、软件的易用性、灵活性和客户化程度，对业务的全局性和局部性信息的需求（战略、战术需求）。

商业智能的用户类型、角色、需求、分析方法对照如表 6-1 所示。

表 6-1 商业智能用户对照表

用户类型	角色	需求	分析方法
中下级经理和业务人员	固定报表读者	需要阅读数据仓库定时或按条件产生的固定报表	固定查询、产生报表
	信息浏览者	根据不同的业务需求，通过建立简单的查询，进行分析，产生动态报表	自助查询、动态报表
高层决策者	管理（或称领导）信息系统用户	了解宏观业务状况，并能迅速定位到反映问题原因的微观细节	了解反映业务状况的关键性能指标，多维分析，穿透和钻取
数据分析专家	数据分析用户	根据不同的业务要求，建立自己的数据模型进行随机查询；通过多维分析，进行各种高级查询和报表	多维分析、趋势分析、对比分析、排名分析、意外分析、原因影响分析、假设分析（What if）
	数据挖掘用户	根据现有的数据情况，动态构建或修改模型，进行预测分析、数据挖掘等深层次操作	统计分析（预测、假设检验等）；数据挖掘(估计、预测、分类、聚类分析等)

6.1.4 商业智能的体系结构

把商业智能系统工作的过程进行技术上的抽象，可以把商业智能的体系结构进行分层，如图 6-5 所示，根据数据的不同形态，整个体系被划分为 4 个大的层面，并根据数据的处理和应用过程再细分成 7 个环节，这些环节通过密切的协助完成商业智能的功能。

数据从数据源经过抽取、转换、装载过程加载到中央数据仓库，再从数据仓库经过分类加工放到数据集市（Data Market，DM），或将数据集市中的数据进一步存放到多维数据库，这都属于数据组织的问题，从中间层到终端用户或从多维数据库到终端用户可将其划归为前端应用实现的问题。而贯穿整个体系数据处理环节的，是系统的流程调度控制和元数据管理。

1．数据源

数据源可以是企业日常运作积累下来的各类的业务数据，也可以是外部的数据。这些数据在存放方式、存放格式、存放地点上可能是多种多样的，这就要求了数据仓库的体系结构必须能处理这种多样性带来的种种问题，如访问多种技术平台下，多种类型的数据库管理系统（DataBase Management System，DBMS）内的数据，并解决由于数据远程迁移所带来的完整性和安全性的问题。

2．ETL

数据抽取、转换和装载完成如下任务：从源数据抽取数据、进行一定的变换、装载

到数据仓库。在上述过程中，需要进行如下数据处理。

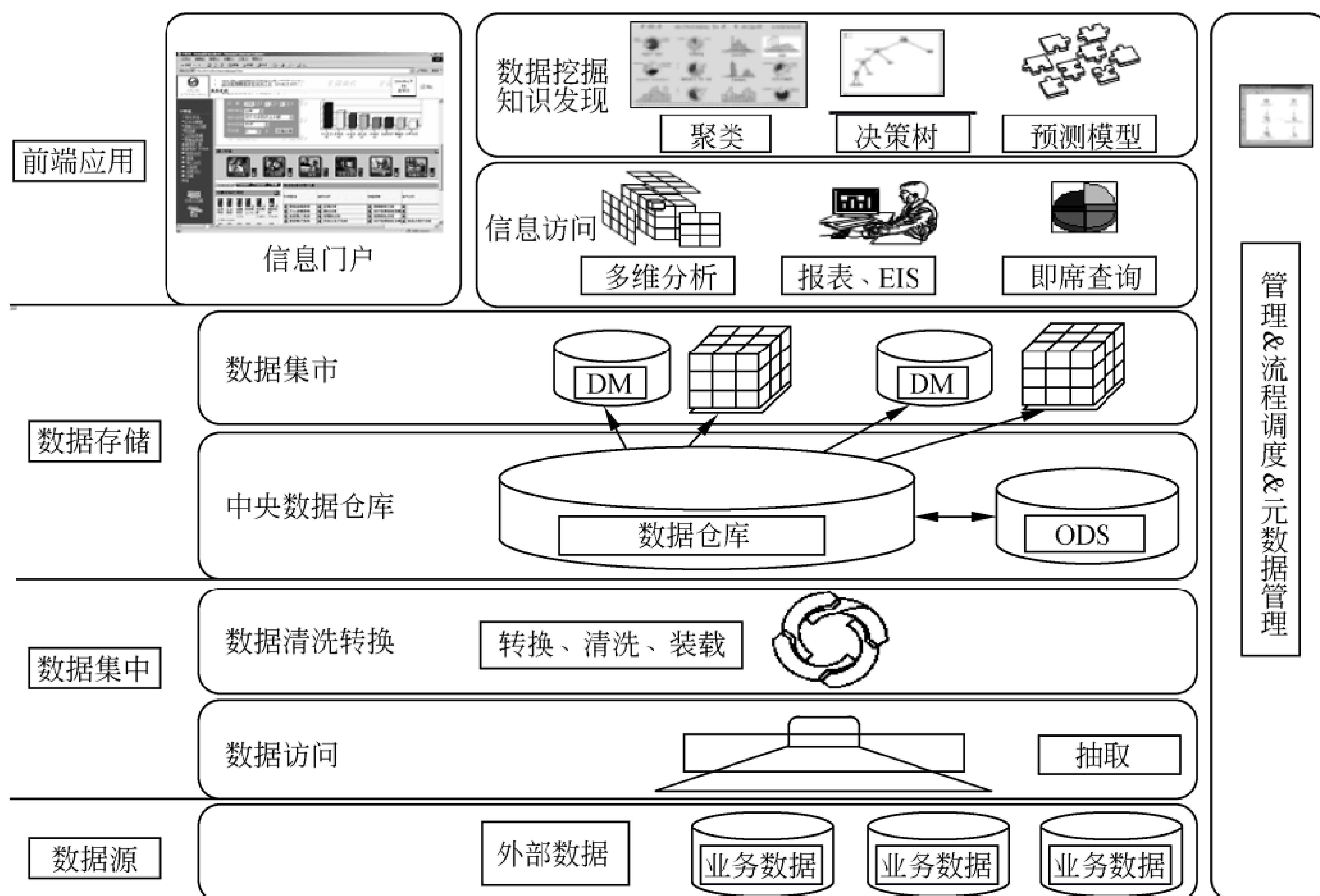


图 6-5　商业智能解决方案体系结构图

（1）简单变换：是数据变换最简单的形式，一次只针对一个字段，而不是考虑相关字段的值。主要有数据类型的转换、日期/时间的格式转换、字段解码等。

（2）清洁和刷洗：目的是为了保证前后一致地格式化和使用某一字段或相关的字段群。清洁和刷洗是两个可以互换的术语，指的是比简单变换更为复杂的一种变换。在这种变换中，要检查的是字段和字段组中的实际内容而不仅是存储格式。一种检查是检查数据字段值的有效值，它指的是检验一个字段的有效值以保证它落在预期的范围之内，通常是数字范围和日期范围。数据刷洗的另一主要类型是重新格式化某些类型的数据，这种方法适用于可以用许多不同方式存储在不同数据来源中的信息，必须在数据仓库中把这类信息转换成一种统一的表示方式。

（3）集成：要把从全然不同来源的数据结合在一起，真正的困难在于将其集成一个紧密结合的数据模型。这些数据来源往往遵守的不是同一套业务规则，在生成新数据时，必须考虑到这一差异。

（4）聚集和概括：大多数数据仓库都要用到数据的某种聚集和概括。这通常有助于将某一实例的数目减少到易于驾驭的水平，也有助于预先计算出广泛的概括数字，以使每个查询不必计算它们。概括是指按照一个和几个业务维将相近的数值加在一起，聚集

是将不同业务元素加在一起或为一个公共总数，在数据仓库中它们是以相同的方式进行的。

3．操作型数据存储区

操作型数据存储区（Operational Data Store，ODS）是为了弥补业务系统和数据仓库之间的数据同步差距而提出的，要解决的是这种问题：“对一个特定的业务流程来说，我怎么才能提供最新的、跨功能部门之间的信息”。例如，对客户服务人员，他需要销售、库存、市场和研发等各部门的最新数据，而这些数据原来是分散在不同部门的不同应用系统的。如果通过数据仓库来实现数据集成，则实时性难以保证，或建设成本很高。

ODS 是数据仓库体系结构中的一个可选部分，ODS 具备数据仓库的部分特征和联机事务处理（On-Line Transaction Processing，OLTP）系统的部分特征，它是“面向主题的、集成的、当前或接近当前的、不断变化的”数据，与数据仓库类似，ODS 也是面向主题的、集成的，但是其最大特点是数据是可更新的，甚至由业务系统通过触发器直接更新。因此，ODS 是业务系统和 DW 之间更偏向业务系统的数据存储区域。

有关 ODS 的更加详细的知识，将在本书的第 7 章进行介绍。

4．数据仓库

数据仓库的一个目的就是把企业的信息访问基础从一种非结构化的或发展中的环境改变成一种结构化或规划良好的环境。关于数据仓库的详细描述，将在 6.2 节和 6.3 节中进行介绍。

5．数据集市

简单地把数据集市理解成数据仓库的一部分是不对的，因为两者虽然在数据上有非常密切的联系，而定位上却是不同的，关于数据集市的详细描述，将在 6.2.5 节中进行介绍。

6．前端应用

商业智能的前端应用是建立数据仓库的目的，如果没有前端应用，DW 就失去了意义。另一方面，由于最终用户的要求多种多样，不可能用同一个界面满足所有用户的信息查询要求，必须根据用户的特点提供不同的界面。最终用户对 DW 中的数据的访问方式包括即席查询、报表、OLAP、数据挖掘和 EIS 等，用户可以通过浏览器或其他前端工具访问 DW 中的数据。

（1）即席查询和报表

即席查询（Adhocery Query）和报表是 BI 系统提供给业务人员最基本的信息访问能力，满足他们日常报表和随时获取业务信息的需要。不同的业务人员（如销售、市场、财务等）有着自己独特的分析要求，且这种要求需根据业务的需要不断变化。在传统的技术条件下，由于种种理由业务人员实质上是不能直接接触到存在计算机内的数据，如果业务人员需要对一段时间的业务汇总数据，往往只能提出要求，由 IT 人员编写相应的程序把数据库中的数据读出来生成报表，再通过批处理打印的方法将结果交给业务人员，

这种方法已经日益不能满足业务人员对动态、及时及个性化信息的要求。同时，这种对IT 人员过多的依赖消耗太多的 IT 资源，增加了管理和运作的成本。因此必须在 IT 与业务用户之间正确地划分权限，既能方便用户自助查询，又能保证 IT 的统一管理的即席查询和报表功能是商业智能系统必须具备的功能。

用户界面的友好性是一直以来商业智能的前端工具的一个着力点，用户可通过简单的鼠标点击、拖拉等操作就可以完成复杂的查询功能，可以在一个文档中包含来自多个数据源的数据，可以完成各种统计、排序、分组、计算工作，可以通过限制字段的值对结果进行过滤，可以通过高亮度显示突出特殊的结果集。而用传统的方式下，构造复杂的结构化查询语言（Structured Query Language，SQL）查询语句，各种复杂的统计和处理，结果的输出这些都需要编写大量程序代码来实现，而报表用户任何轻微的改动会给IT 技术人员带来的繁复的编程工作。

可以说引入这些为最终用户设计的数据查询和报表工具，一方面让最终用户真正拥有了自由查询自己需要信息的能力，另一方面，把信息的查询直接还给最终用户，IT 人员就可以把更多的精力放在为满足大的方面业务需求的数据后台整合工作上，对于 IT 人员和业务人员来说双重的解放。

即席查询和报表工具是集成查询和报表的解决方案，具有易于使用和二次开发的特点。

（2）OLAP

OLAP 又称为多维分析，管理人员往往希望从不同的角度观察数据来审视业务情况，比如从时间、地域、产品、客户等来看收入、利润、支出等业务统计数字。每一个分析的角度可以叫做一个维，因此，把多角度分析方式称为多维分析。以前，每一个分析的角度需要制作一张报表。多维分析工具的主要功能，是根据用户常用的多种分析角度，事先做好汇总和计算，以便在查询时能尽快访问到所要的汇总数字，并快速地从一个维度转变到另一维度继续观察数据。

图 6-6 直观地表示了一个贷款分析模型所能实现的可能的分析角度（维度）和数据层次（粒度），如图 6-7 所示。

很明显，这个简单的分析模型已经包含了 8×8×4×4=1024 种不同角度不同层次对授信金额和贷款金额的统计分析，下面看看一些多维分析的操作。

① 切片和切块操作（Slice and Dice）。在多维数据结构中，按二维进行切片，按三维进行切块，可得到所需要的数据。如在“贷款银行、贷款质量、时间”三维立方体中进行切块和切片，可得到各贷款银行、各种贷款的统计情况。每次都是沿其中一维进行分割称为分片，每次沿多维进行的分片称为分块。图 6-8 是 2004 年 4 月份所有贷款情况的切片，而图 6-9 是所有不良贷款情况的切片。

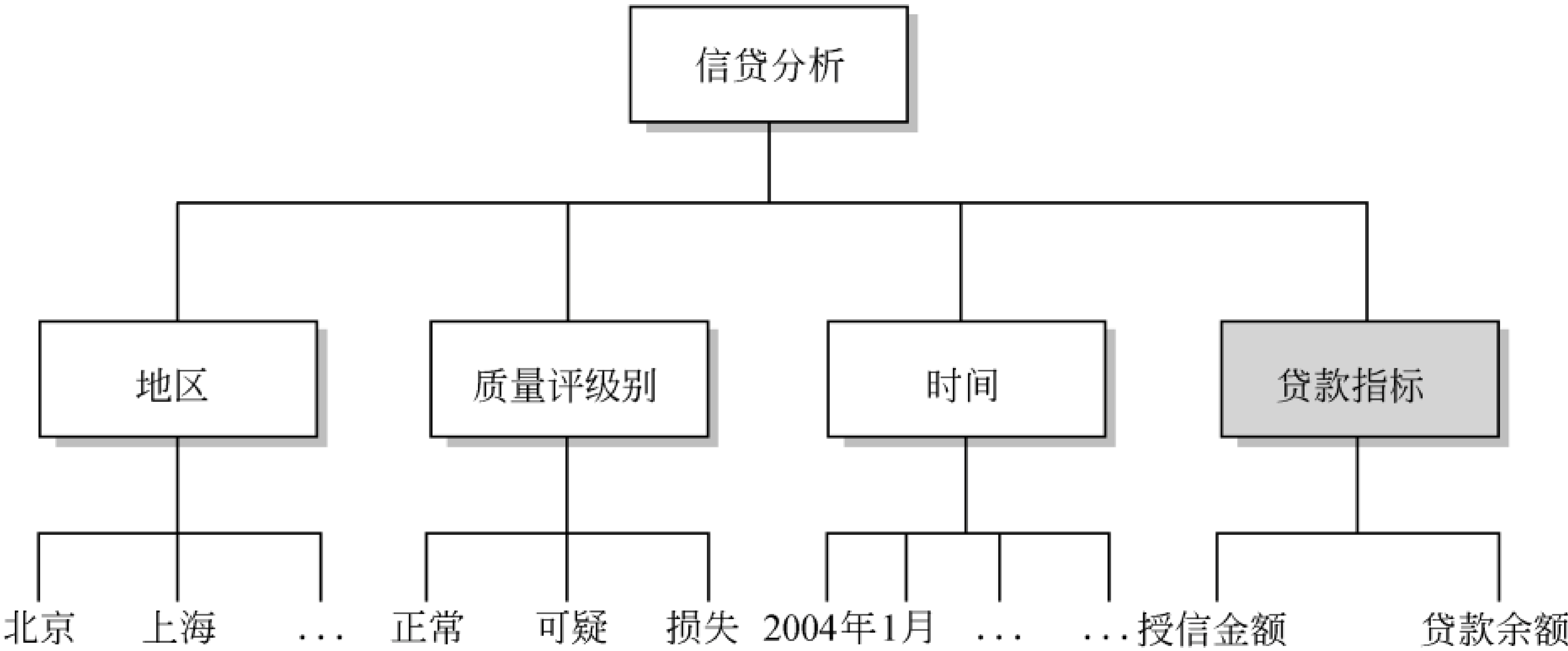


图 6-6　信贷分析模型

维度 →

粒度 ↓

时间	贷款银行	区域	贷款质量
年	商业银行总行	省	正常/不良
季度	省级分行	市	五级分类
月	市分行		

度量指标（事实）：授信金额、贷款金额

图 6-7　贷款分析的角度和层次

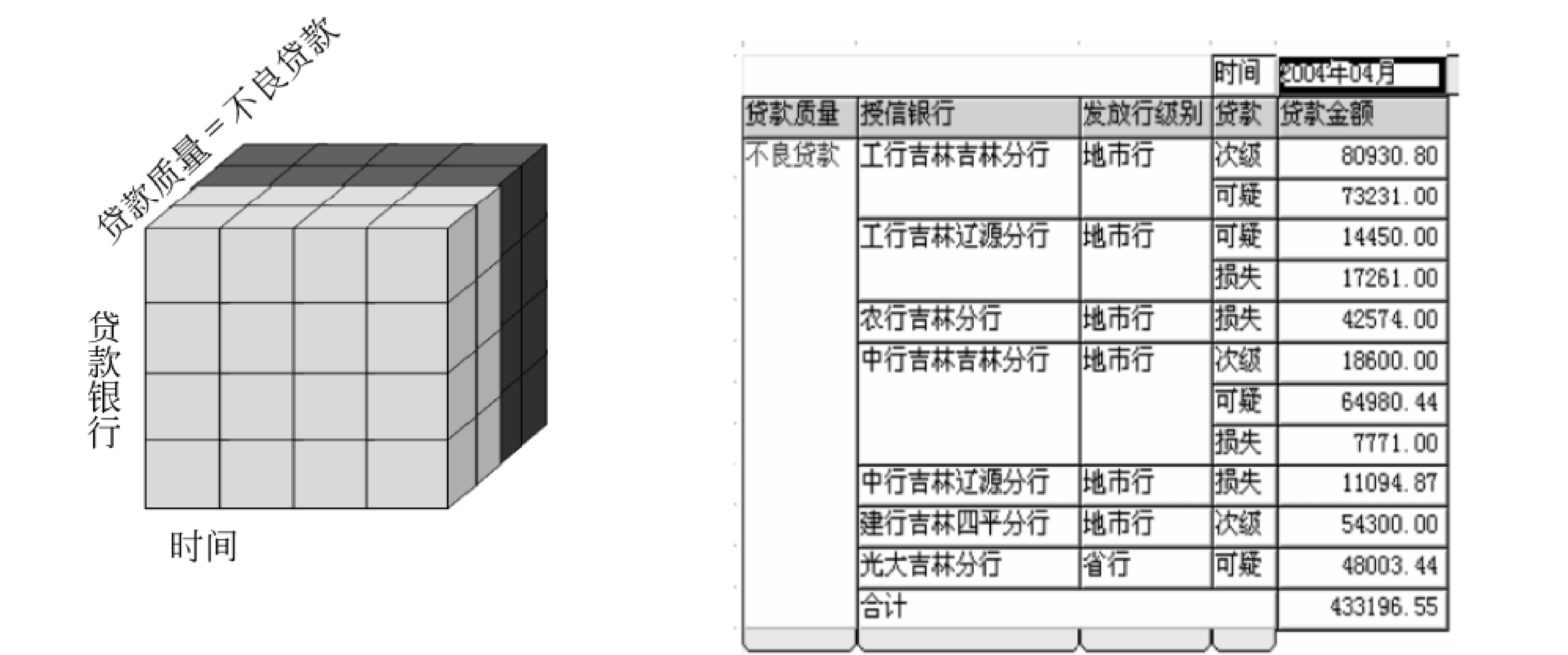
贷款质量

贷款银行

时间=2004年4月

时间	授信银行	发放行级别	贷款	贷款质量	贷款金额
2004年04月	工行吉林吉林分行	地市行	正常	正常贷款	273684.00
			关注	正常贷款	114924.00
			次级	不良贷款	80930.80
			可疑	不良贷款	73231.00
	工行吉林四平分行	地市行	正常	正常贷款	81309.00
	工行吉林辽源分行	地市行	正常	正常贷款	82573.48
			可疑	不良贷款	14450.00
			损失	不良贷款	17261.00
	农行吉林分行	地市行	正常	正常贷款	95106.50
			损失	不良贷款	42574.00
	农行吉林吉林分行	地市行	正常	正常贷款	11260.00
	农行吉林辽源分行	地市行	正常	正常贷款	17080.00
			关注	正常贷款	10120.00
	中行吉林吉林分行	地市行	正常	正常贷款	57391.93
			关注	正常贷款	90815.89
			次级	不良贷款	18600.00
			可疑	不良贷款	64980.44
			损失	不良贷款	7771.00
	中行吉林四平分行	地市行	关注	正常贷款	16865.00
	中行吉林辽源分行	地市行	关注	正常贷款	26494.81
			损失	不良贷款	11094.87
	建行吉林吉林分行	地市行	正常	正常贷款	50000.00
			关注	正常贷款	20000.00
	建行吉林四平分行	地市行	正常	正常贷款	1000.00
			关注	正常贷款	126900.00
			次级	不良贷款	54300.00
	交行吉林吉林分行	地市行	正常	正常贷款	15750.00
	光大吉林分行	省行	正常	正常贷款	66570.00
			关注	正常贷款	42600.00
			可疑	不良贷款	48003.44
	合计				1633641.16

图 6-8　切片一：2004 年 4 月份所有贷款情况



			时间	2004年04月
贷款质量	授信银行	发放行级别	贷款	贷款金额
不良贷款	工行吉林吉林分行	地市行	次级	80930.80
			可疑	73231.00
	工行吉林辽源分行	地市行	可疑	14450.00
			损失	17261.00
	农行吉林分行	地市行	损失	42574.00
	中行吉林吉林分行	地市行	次级	18600.00
			可疑	64980.44
			损失	7771.00
	中行吉林辽源分行	地市行	损失	11094.87
	建行吉林四平分行	地市行	次级	54300.00
	光大吉林分行	省行	可疑	48003.44
	合计			433196.55

图 6-9　切片二：所有不良贷款情况

② 钻取操作（Drill）。钻取包括向下钻取（Drill-down）和向上钻取（Drill-up）/上卷（Roll-up）操作，钻取的深度与维所划分的层次相对应。图 6-10 是一个钻取的示意图。

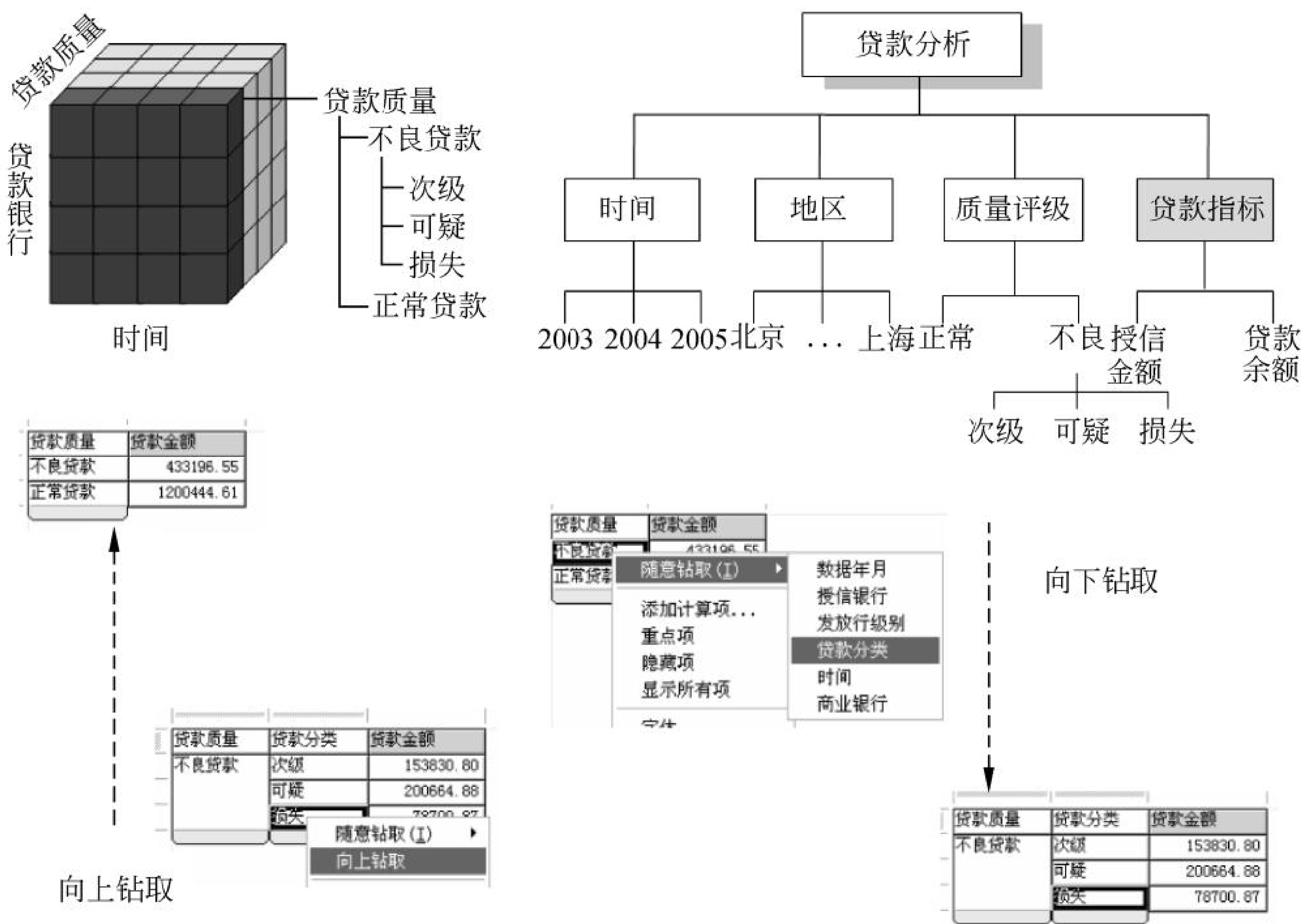
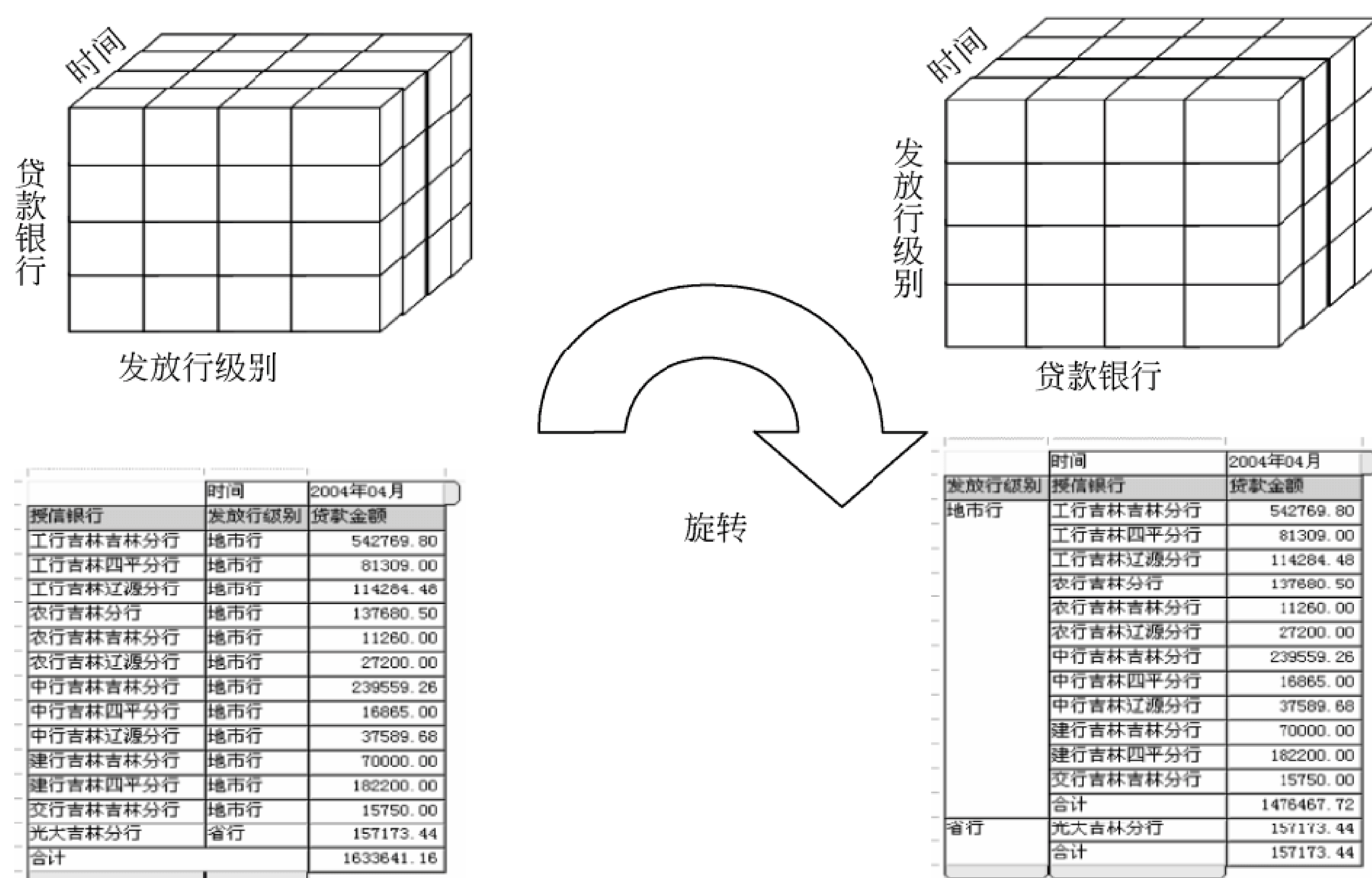


图 6-10　钻取示意图

③ 旋转（Rotate）/转轴（Pivot）操作。通过旋转可以得到不同视角的数据，如图

6-11 所示。



	时间	2004年04月
授信银行	发放行级别	贷款金额
工行吉林吉林分行	地市行	542769.80
工行吉林四平分行	地市行	81309.00
工行吉林辽源分行	地市行	114264.46
农行吉林分行	地市行	137680.50
农行吉林吉林分行	地市行	11260.00
农行吉林辽源分行	地市行	27200.00
中行吉林吉林分行	地市行	239559.26
中行吉林四平分行	地市行	16865.00
中行吉林辽源分行	地市行	37589.68
建行吉林吉林分行	地市行	70000.00
建行吉林四平分行	地市行	182200.00
交行吉林吉林分行	地市行	15750.00
光大吉林分行	省行	157173.44
合计		1633641.16

	时间	2004年04月
发放行级别	授信银行	贷款金额
地市行	工行吉林吉林分行	542769.80
	工行吉林四平分行	81309.00
	工行吉林辽源分行	114284.48
	农行吉林分行	137680.50
	农行吉林吉林分行	11260.00
	农行吉林辽源分行	27200.00
	中行吉林吉林分行	239559.26
	中行吉林四平分行	16865.00
	中行吉林辽源分行	37589.68
	建行吉林吉林分行	70000.00
	建行吉林四平分行	182200.00
	交行吉林吉林分行	15750.00
	合计	1476467.72
省行	光大吉林分行	157173.44
	合计	157173.44

图 6-11　旋转示意图

从上面的多维操作，可以看到，通过对数据观察角度的变换，可以更容易全面而深入地了解到一些关于“为什么”的信息。

（3）EIS

EIS 是针对管理人员的需要，整合上述各种功能控制的前端应用。通过 EIS，将管理人员所需的决策信息按需集成到统一的界面中，帮助他们能够快速、直接地访问信息。与其他信息查询方式相比，EIS 更强调与用户的交互能力，除了以多种形式展示数据内容外，EIS 还可以以下拉列表、按钮、选项、图标等多种屏幕控件响应用户的操作，并能通过对界面的美工增强对用户的亲和力。

6.2　数据仓库技术

从 6.1 节的描述中已经知道，数据仓库的设计是商业智能应用中十分重要的一环，数据仓库是商业智能应用最基本的环境，OLAP 分析、报表和查询、数据挖掘等商业智能应用都需要数据仓库作为共同的基础。

数据仓库来源于数据库，其本身也是由数据库管理系统来管理的，但是它的结构、功能和设计都和传统的数据库设计方法不同，本节将会对数据仓库技术的知识和设计方法进行深入学习，探究创建数据仓库的理论和方法。

6.2.1　操作型数据和分析型数据

传统的企业信息化实现的是用计算机信息处理代替人工信息处理，主要解决的是业务上的数据流问题。来看一个简单的例子。在银行中，一般都有存款、贷款、信用卡、代理业务等多种业务系统，它们都是支持相关业务处理而设计的交易处理系统，系统主要任务是完成日常业务交易过程中的数据处理，这种操作型系统的使用人员通常是企业的具体操作人员，处理的数据通常也是企业业务中的细节信息，比如具体的一笔业务。

针对操作型数据处理的 OLTP 系统，总是按照业务应用来建立它的数据模型，换言之，业务处理系统是面向操作应用来设计的，联机业务处理系统更是面向交易来设计，存储操作型数据的数据库在设计的时候主要是围绕性能和完整性方面，而每个交易涉及的数据往往只是记录的层面，数据库设计主要考虑对并行更新的支持比较多，并不需要考虑为全局查询做优化。另外，企业针对不同业务往往可能有多个操作型的系统，这些系统开发的时候都是独立进行的，相互之间没有什么数据联系，各系统之间对实际业务中相同的信息在数据上是冗余的，而在不同的系统表达方式和数据内容上很可能不一致，甚至项目矛盾。例如，每个系统中都会有存放客户部分信息的数据，信息分布的零碎和冗余，使决策者很难从这些业务系统中直接获取全面的信息。

分析型系统的使用人员通常是企业的中高层管理者，或是从事数据分析的分析师，他们关注的更多是企业宏观的信息而非具体的细节，其使用数据的目的是为企业的决策者提供决策支持。分析型系统直接在操作型系统中提取数据会遇到下面一些问题：

（1）操作型数据之间往往需要复杂的关系来保持快捷性、一致性和实时性，要将其直接用于分析，需要创建很复杂的特殊查询语句，这项工作的技术复杂度明显不符合数据分析的用户群的需要。

（2）在事务处理系统中进行数据分析，由于短时间需要查询大量的数据，一方面会明显影响事务处理系统的处理速度和性能，另一方面也会由于响应时间过慢而影响分析的效率。

（3）在进行预测、决策时需要大量全面、正确的集成数据，所有这些集成数据不仅包括企业内部的数据，还包括企业外部的数据，如行业信息、竞争对手信息等。而操作型数据仅仅保存与本业务相关的信息，缺少与决策相关的集成数据尤其是企业外部数据。

（4）历史数据问题。供决策分析的数据一般是历史数据，而操作型数据库一般只保留当前或近期的数据信息。

以上诸多问题的存在，导致企业或其他组织机构无法直接使用现有的业务系统中存储操作型数据的传统数据库系统以满足预测、决策分析的需要。因此，预测、决策分析需要一个能够不受传统事务处理的约束，高效率处理决策分析数据的支持环境，数据仓库就是满足分析型系统要求的数据存储和数据组织环境。

6.2.2 与传统数据库的区别

功能决定结构，承接上一节的讨论，OLAP 系统和 OLTP 系统从本质上是不同的，数据仓库虽然是从传统数据库系统发展而来，但是两者还是存在着诸多差异，例如，从数据存储的内容看，数据库只存放当前值，而数据仓库则存放历史值；数据仓库数据的目标是面向业务操作人员的，为业务处理人员提供数据处理的支持，而数据仓库则是面向中高层管理人员的，为其提供决策支持。表 6-2 详细说明了数据仓库与传统数据库的区别。

表 6-2 数据仓库与传统数据库的比较

比 较 项 目	数 据 库	数 据 仓 库
数据内容	当前值	历史的、归档的、归纳的、计算的数据（处理过的）
数据目标	面向业务操作程序、重复操作	面向主体域，分析应用
数据特性	动态变化、更新	静态、不能直接更新，只能定时添加、更新
数据结构	高度结构化、复杂，适合操作计算	简单、适合分析
使用频率	高	低
数据访问量	每个事务一般只访问少量记录	每个事务一般访问大量记录
对响应时间的要求	计时单位小，如秒	计时单位相对较大，除了秒，还有分钟、小时

6.2.3 数据仓库的特点

数据仓库的特点可以从数据仓库的定义来理解，目前，数据仓库一词尚没有一个统一的定义，著名的数据仓库专家 W.H.Inmon 在其著作 *Building the Data Warehouse* 一书中给予如下描述：“数据仓库是一个面向主题的（Subject Oriented）、集成的（Integrated）、非易失的（Non-Volatile）、且随时间变化的（Time Variant）的数据集合，用于支持管理决策”。他指出了数据仓库的 4 个最重要的特征。

（1）面向主题的。操作型数据库的数据组织面向事务处理任务（面向应用），各个业务系统之间各自分离，而数据仓库中的数据是按照一定的主题域进行组织，如图 6-12 所示。主题是一个抽象的概念，是指用户使用数据仓库进行决策时所关心的重点方面，一个主题通常与多个操作型系统的数据相关。例如，一个银行的事务处理（应用问题）包括存款业务、信用卡业务、贷款业务和代理业务等，而银行的主要主题范围是客户、产品和渠道等。

（2）集成的。在数据仓库的所有特性中，这是最重要的。面向事务处理的操作型数据库通常与某些特定的应用相关，数据库之间相互独立，并且往往是异构的。而数据仓

库中的数据是在对原有分散的数据库数据抽取、清理的基础上经过系统加工、汇总和整理得到的，必须消除源数据中的不一致性，以保证数据仓库内的信息是关于整个企业的一致的全局信息。下面，通过一个例子来说明，当数据由面向事务处理的操作型数据向数据仓库传送时所进行的集成。假设有 4 个不同的应用系统，系统中对人的性别的标识如表 6-3 所示。

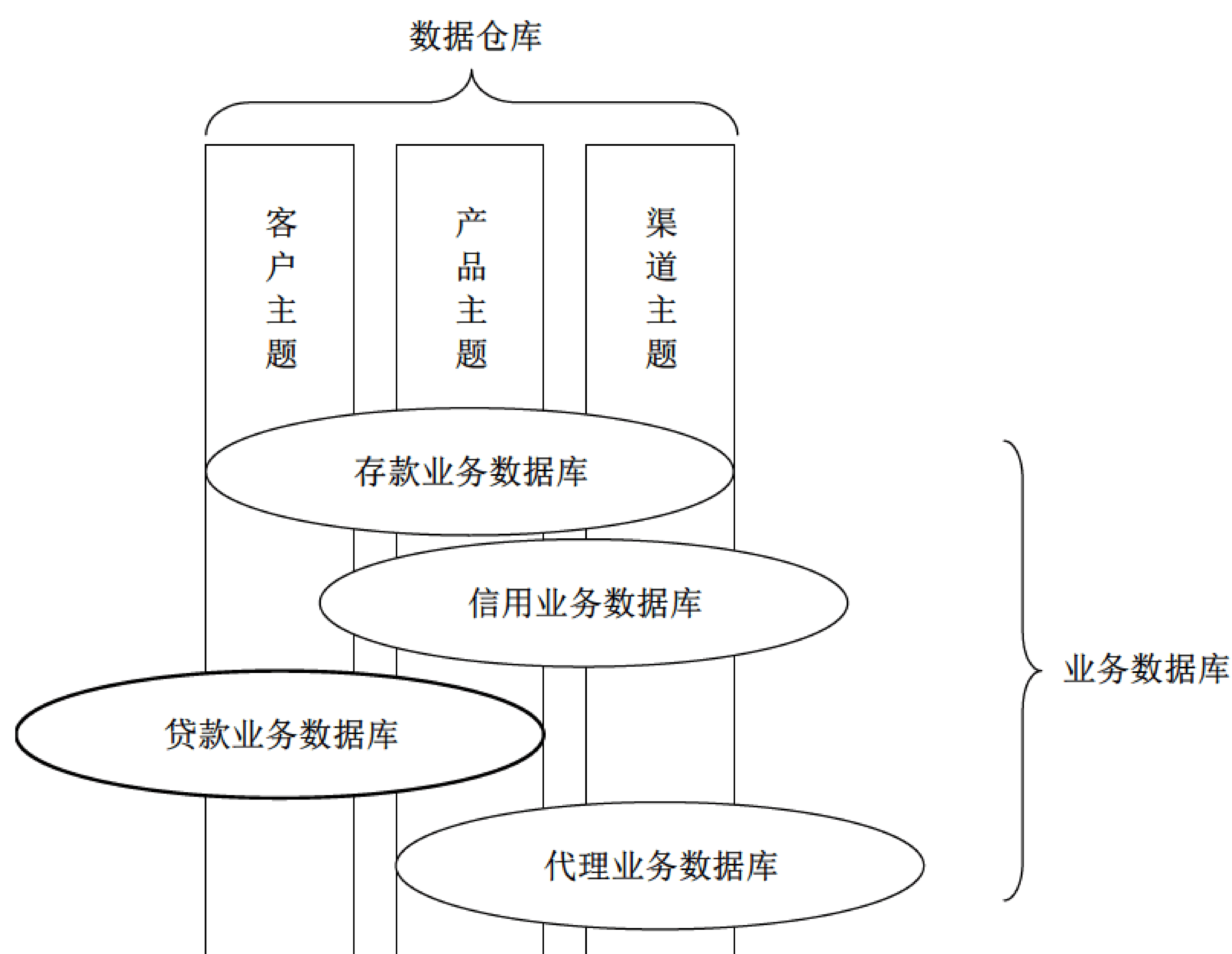


图 6-12　数据仓库面向主题的特性

表 6-3　对性别的表示

	男　　性	女　　性		男　　性	女　　性
系统 A	男	女	系统 C	1	0
系统 B	m	f	系统 D	M	F

那么，在将 4 个系统的性别信息向数据仓库导入时就涉及到集成问题，例如，可以统一将性别信息表示为 m，f。

（3）非易失的（相对稳定性）。操作型数据库中的数据通常实时更新，数据根据需要及时发生变化。数据仓库的数据主要供企业决策分析之用，所涉及的数据操作主要是数据查询，一旦某个数据进入数据仓库以后，一般情况下将被长期保留，也就是数据仓库中一般有大量的查询操作，但修改和删除操作很少，通常只需要定期的加载、刷新。

图 6-13 说明了操作型数据环境下，是正规地一次访问和处理一个记录，可以对数据进行更新（修改、插入、删除）。但数据仓库中的数据却表现出不同的特性，数据通常是

被一起载入和访问的，而且在数据仓库环境中并不进行一般意义上的数据更新操作。

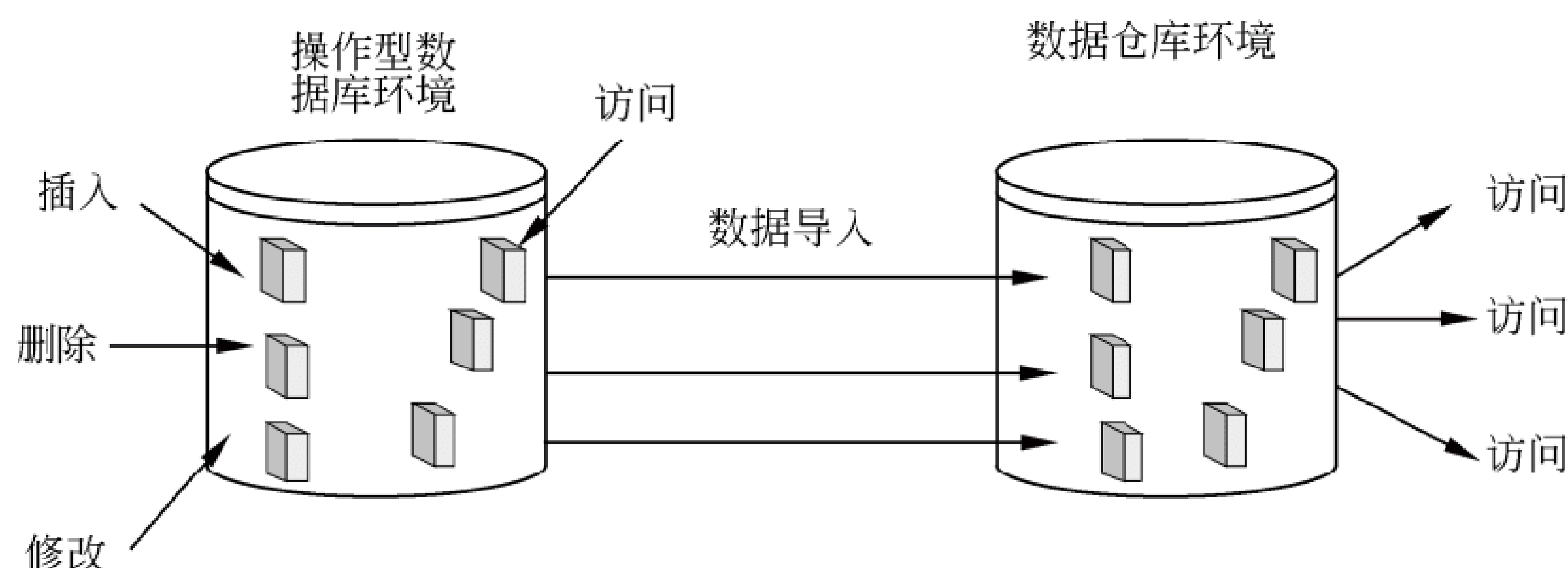


图 6-13　数据仓库的相对稳定性

（4）随时间变化的。反映历史变化或说是随着历史变化。操作型数据库主要关心当前某一个时间段内的数据，而数据仓库中的数据通常包含历史信息，系统记录了企业从过去某一时点（如开始应用数据仓库的时点）到目前的各个阶段的信息，通过这些信息，可以对企业的发展历程和未来趋势做出定量分析和预测。数据仓库的反映历史变化的属性主要表现在以下三个方面：

- 数据仓库中的数据时间期限要远远长于传统操作型数据系统中的数据时间期限，传统操作型数据系统中的数据时间期限可能为数十天或数个月，数据仓库中的数据时间期限往往为数年甚至几十年。
- 传统操作型数据系统中的数据含有“当前值”的数据，这些数据在访问时是有效的，当然数据的当前值也能被更新，但数据仓库中的数据仅仅是一系列某一时刻（可能是传统操作型数据系统）生成的复杂的快照。
- 传统操作型数据系统中可能包含也可能不包含时间元素，如年、月、日、时、分、秒等，而数据仓库中一定会包含时间元素。

6.2.4　数据仓库的模型设计

目前，主流的关系数据库是采用二维表的形式来表示数据，在设计上使用关系模型理论来建模，建模方法是第三范式（Third Normal Form，3NF），3NF 是关系数据库设计的基础理论，按 3NF 设计的数据库存放业务处理的数据是合适的，同时也体现了操作型数据的信息局部性，如果直接在上面做分析操作，必然需要由很多个表做连接操作才能得到需要的相对完整的信息。

1．星型模型

而数据仓库的结构是要面向多维分析的，并且是按面向主题的方式来组织，星型模式体现的是多维的数据关系，它由一个事实表（Fact Table）和一组维表（Dimension Table）组成。每个维表都有一个维作为主键，所有这些维的主键组合成事实表的主键。事实表

的非主键属性称为度量值（Measure）或事实，它们一般都是数值或其他可以进行计算的数据；而维大都是文字、时间等类型的数据，按这种方式组织好数据就可以按照不同的维（事实表的主键的部分或全部）来对这些度量值数据进行求和、求平均、计数、百分比的聚集计算，这样就可以从不同的角度观察反映所分析的业务状况的数据，图 6-14 给出了一个针对银行贷款业务状况的分析模型的例子。

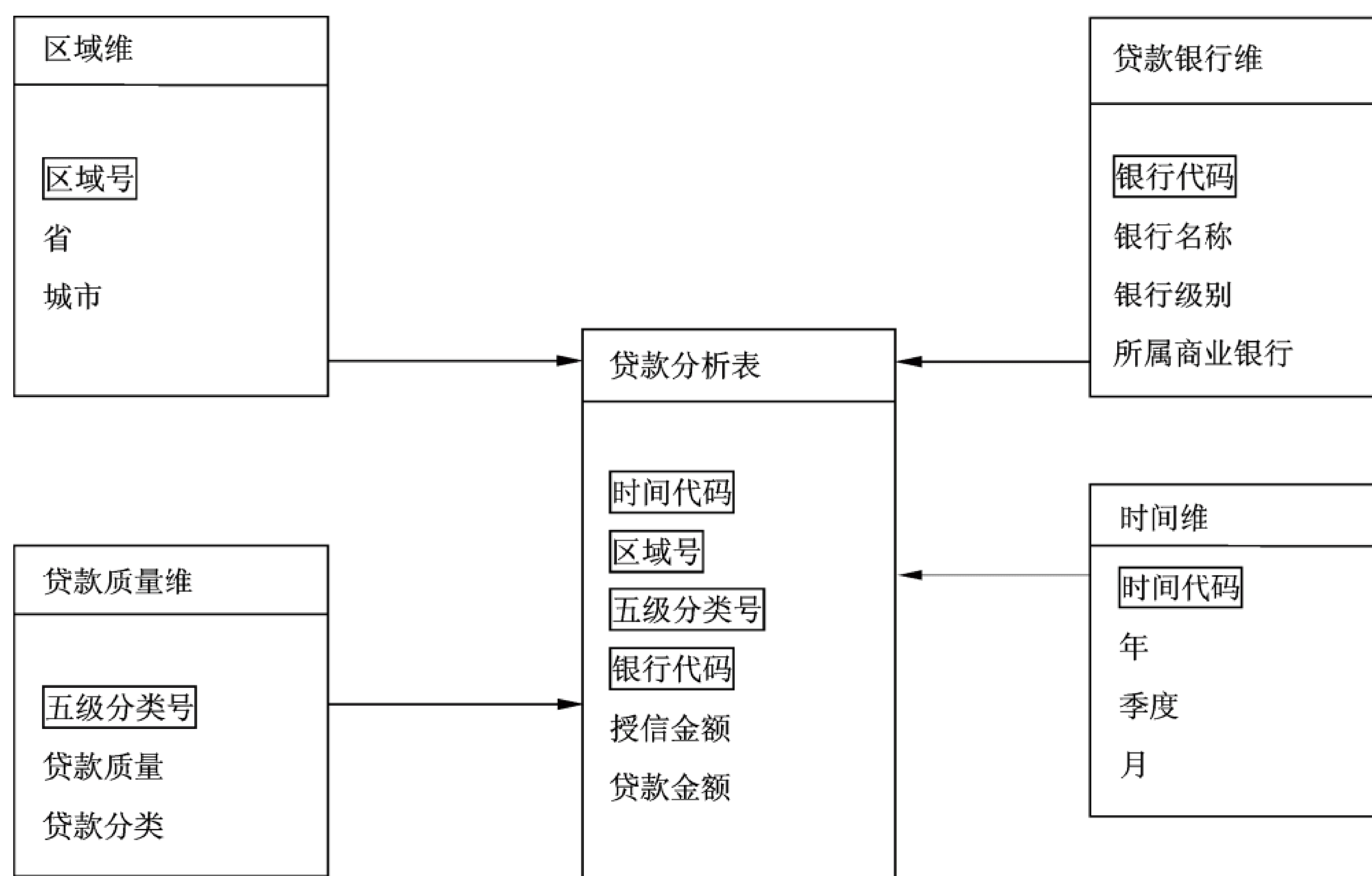


图 6-14　例子：贷款分析星型模型

图 6-14 是银行贷款分析的模型设计，其中加边框的属性为主关键字（Primary Key）。贷款分析表是一个事实表，其中的贷款授信金额，贷款金额（发生额）是需要从各角度观察的数据（度量值），而对这两个数据的观察的角度可以有区域、银行、时间、质量这 4 个方面组合进行，通过这些分析角度的组合，可以对授信金额和贷款余额进行 4×8×4×8 种不同角度的数据统计，由此可以对贷款情况的从多个角度（维）进行分析和观察，贷款分析人员既可以宏观地看到贷款业务的整体情况，又可以微观地观察到具体某一家银行某一天某一类的贷款情况。进行多维分析的时候，维度选择越多数据越细节（划分得更细了），维度选择越少数据越汇总越宏观。

在这个模型中，中间的一个细长的大表（事实表，记录数很多，字段数目却不多）形成主表，周围一组小表（维表，记录数很少）与主表相关联的结构，形态上如图 6-15 所示呈星星的形状，所以被命名为星型结构，星型模型是数据仓库的数据模型与传统关系数据库应用相区分的一个重要特征。

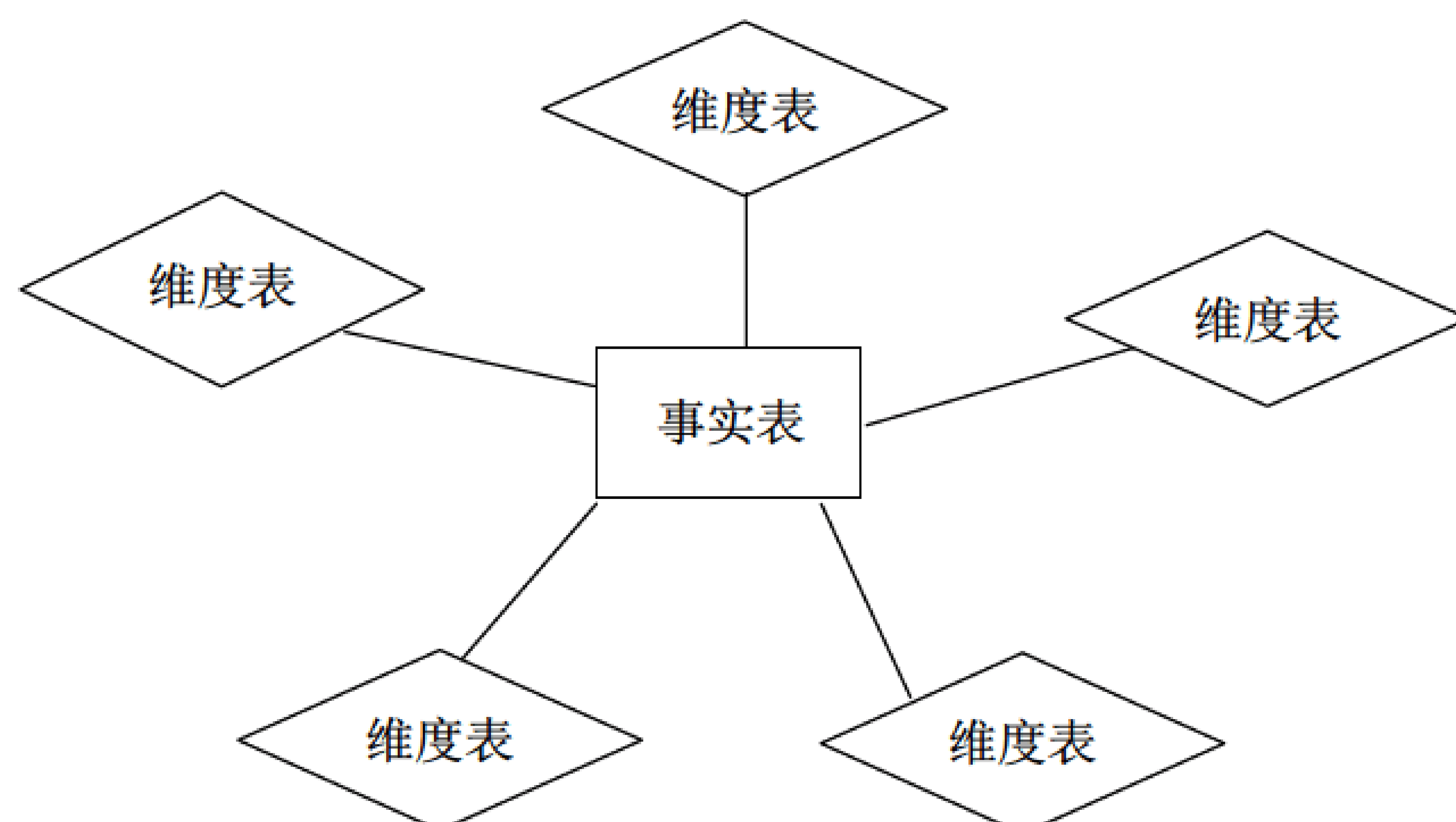


图 6-15　数据仓库典型的逻辑模型形状

星型模型虽然也是一个关系模型，但它不是一个规范化的模型。在星型模型中，维度表被故意非规范化了，使用星型模式主要有两方面的原因。

（1）提高查询效率：同一个主题的主要的数据都存放在庞大的事实表中，只要扫描事实表就可以进行查询，而不必把多个表联接起来。

（2）便于用户理解：星型模式比较直观，通过星型模式，很容易组合出各种查询。

2．雪花模型

雪花模型是对星型模型的一个扩展，每一个维表还可以向外面连接多个详细类别表，如图 6-16。例如，上面的贷款银行维表可以再扩展一个银行级别类表，而形成一个雪花型的结构。而由于数据仓库不必要求考虑满足第三范式，也不必要求考虑避免冗余，可以考虑把详细类别表的字段合并入维表里面。由此可见，在星型模式的基础上拓展而成的雪花型模型，实际上是对分析查询性能和数据仓库容量两方面的平衡。

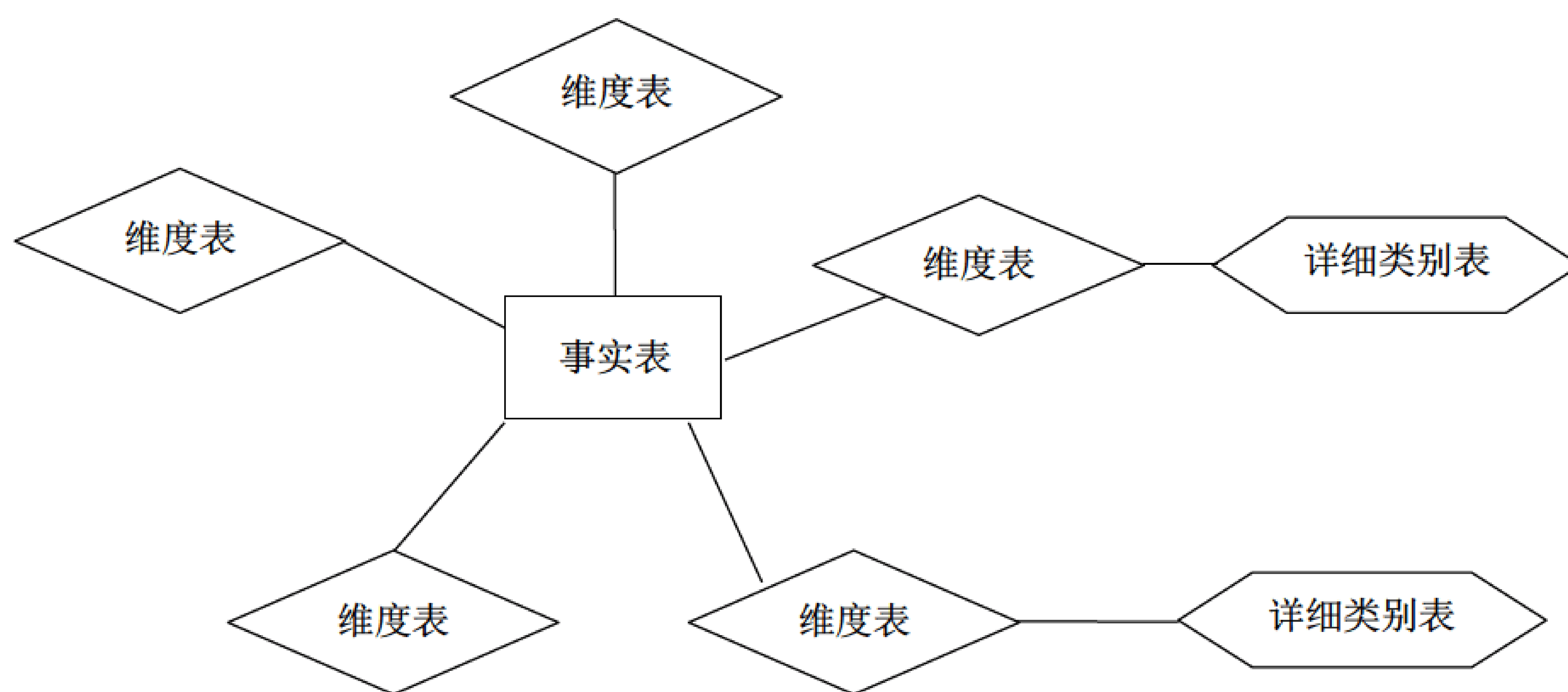


图 6-16　雪花模型示意图

3．星座模型

一个复杂的商业智能应用往往会在数据仓库中存放多个事实表，这时就出现多个事

实表共享一个或多个维表的情况，这就形成了星座模式，例如，可以复用以上贷款分型模型的维表到存款分析模型中，如图 6-17 所示，公共维表设计在数据仓库项目中有普遍意义。

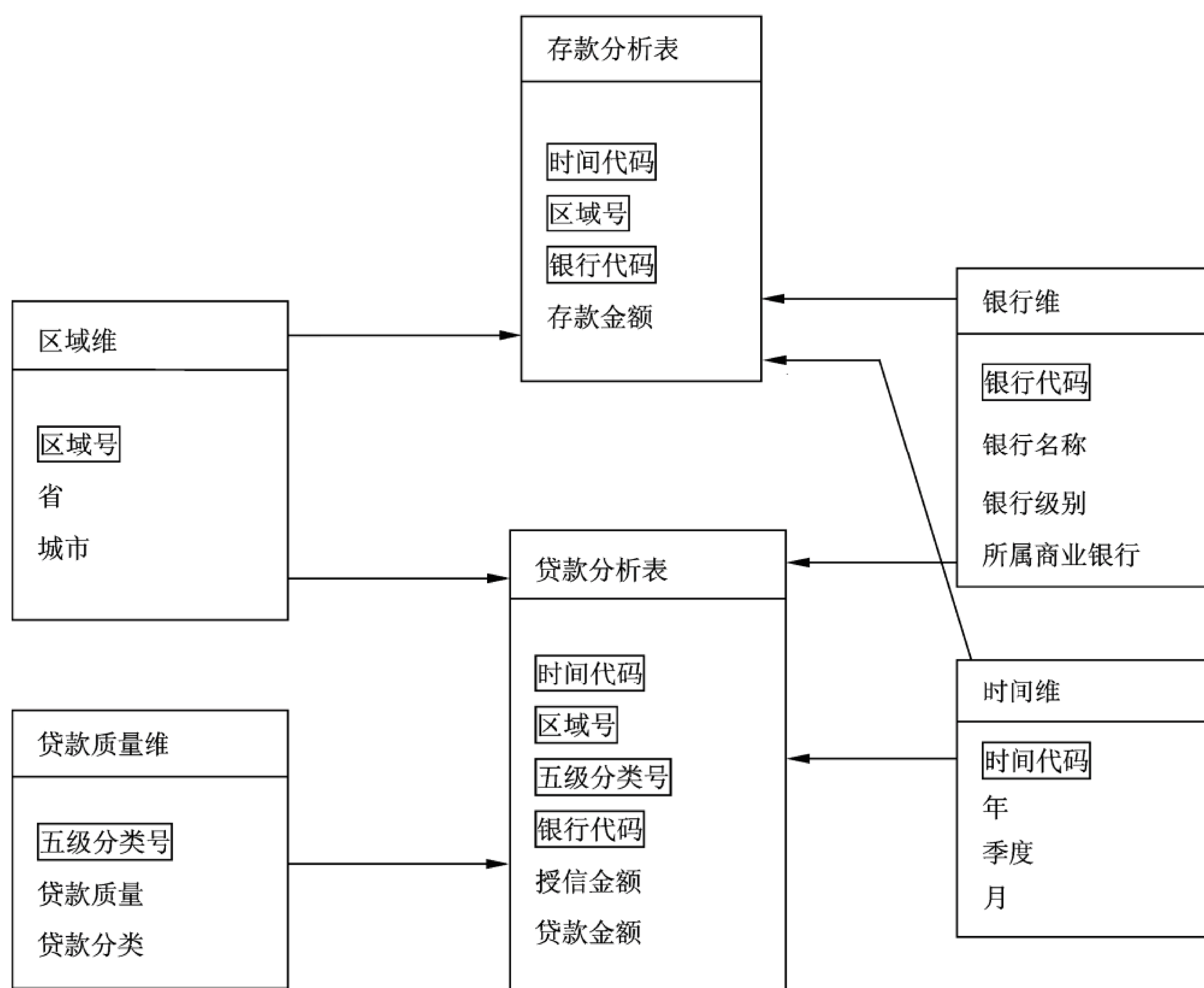


图 6-17　例子：存、贷款分析星座模型

6.2.5　数据集市

简单地把数据集市理解成数据仓库的一部分是不对的，因为两者虽然在数据上有非常密切的联系，而定位上却是不同的。数据仓库所对应的是整个企业层面的整体信息视图，体现决策信息在企业的共性需求。而对于企业内同一个业务概念，由于业务观点的不同导致大家对数据的理解和运用有不同的视角，缺乏针对性的单一个模型并不能都满足这种不同观点的数据需求。例如，客户是现在企业非常重要的一个信息主题，而从产品经理的角度，可能关心的是客户的消费喜好和消费行为，而从财务经理的角度，更多地可能是关心客户的成本和带来的收益，这些不同的数据的使用观点需要不同的数据模型来满足，一般而言，数据仓库可以理解为企业决策信息平台提供总数据支持的，数据

集市可以理解为部门范围级别的决策支持应用而设计的，其数据模型设计和数据组织上更多地服务于一个部门的信息需求。

根据数据来源不同，数据集市有两种类型，分别是独立的数据集市（Independent Data Mart）和从属的数据集市（Dependent Data Mart），如图 6-18 所示。

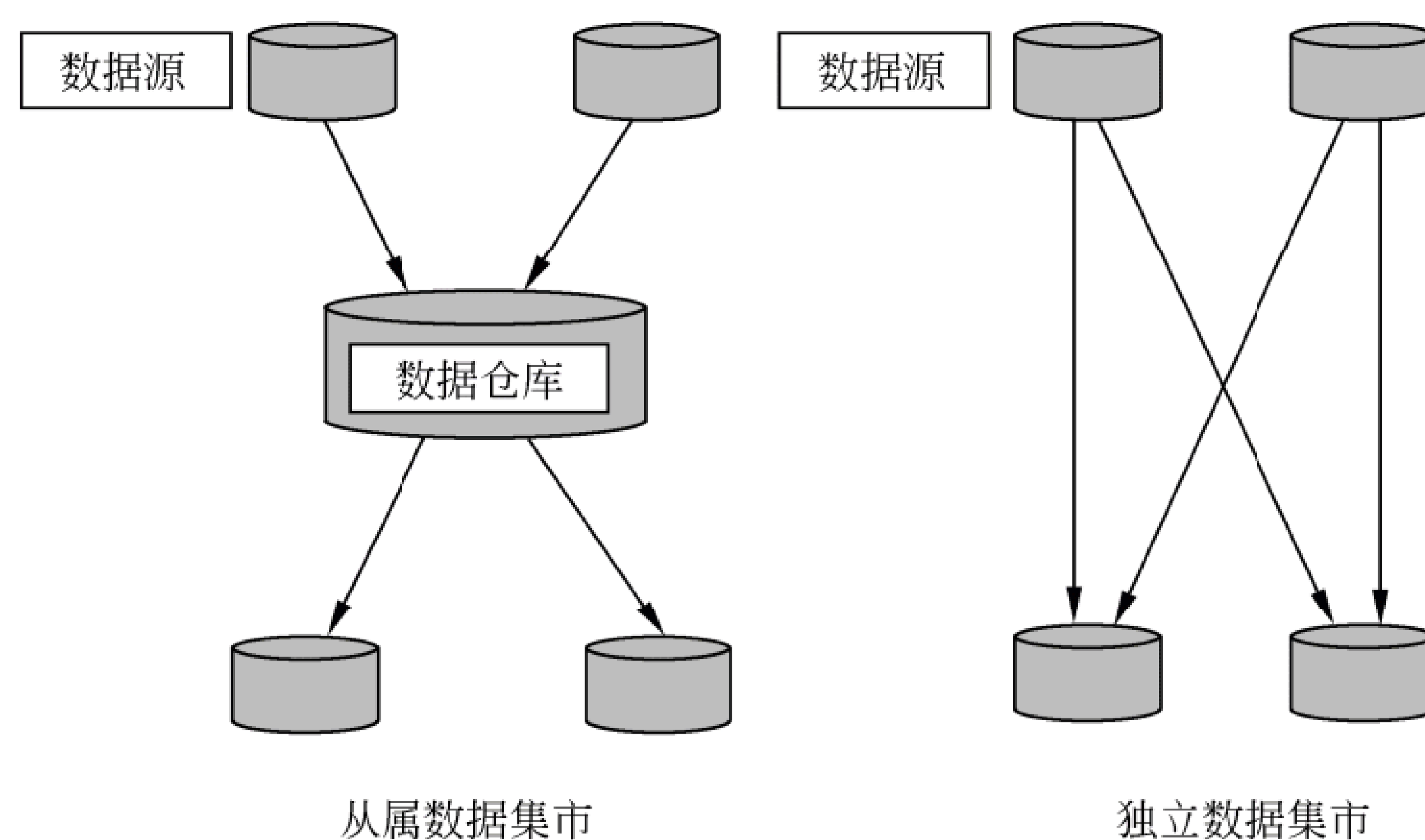


图 6-18 数据集市类型

从属数据集市的数据直接来自于中央数据仓库，这样有利于保持数据的一致性，因为来自同一数据源并且已经经过一致性处理和检验。从属数据集市的作用在于，为一些部门建立数据集市，将需要的数据复制、加工到其中，这样不仅可以提高该部门的访问速度，同时也为能满足该部门的一些特殊的分析需求。

独立数据集市的数据直接来自于业务系统，由于为各个部门都建立了各自的数据集市，而当需要从整体上建立一个数据仓库时，不同数据集市中的数据表达由于各部门的不同特殊需要而有所不同，将这种不一致的数据整合到一个中心数据仓库时，可能会遇到一些困难，比如重新设计、各部门协调等。其优点是建立迅速、价格相对低廉。因此建立独立数据集市往往是由于投资方面的考虑或工期的紧迫，或解决某部门的迫切需要。

表 6-4 总结了从属数据集市和独立数据集市的一些对比。

表 6-4 从属数据集市与独立数据集市对比表

对　　比	优　　点	缺　　点
从属数据集市	保证数据一致性；架构比较理想，可扩展能力强	依赖与中心数据仓库的实施，实施周期长，实施成本高
独立数据集市	实施周期短，实施成本低	没有消除信息分割，可扩展能力弱，后期整合困难

希赛教育专家提示：在设计其他部门的数据集市或中心数据仓库时，要充分考虑现

有数据集市的设计，以避免设计的不一致性而造成后期整合的困难及昂贵的费用。

6.2.6　其他相关概念

本节再介绍一些与数据仓库相关的概念。

1．粒度

粒度是数据仓库的重要概念。粒度可以分为两种形式，第一种粒度是对数据仓库中的数据的汇总程度高低的一个度量，它既影响数据仓库中的数据量的多少，也影响数据仓库所能回答询问信息的种类。在数据仓库中，多维粒度是必不可少的。由于数据仓库的主要作用是多维分析，因而绝大多数查询都基于一定程度的汇总数据之上的，只有极少数查询涉及到细节。

还有一种粒度形式，即样本数据库。它根据给定的采样率从细节数据库中抽取出一个子集。这样样本数据库中的粒度就不是根据汇总程度的不同来划分的，而是有采样率的高低来划分，采样粒度不同的样本数据库可以具有相同的数据汇总程度。

2．聚合

事实表中存放的度量值，根据其实际意义可以分成是可加性的度量值和非可加性的度量值。可加性的度量值指将同一个事实表里面的不同记录的该数值相加得到的结果还有具体意义，譬如上面例子中的贷款金额，将一个季度的 3 个月的数字相加可以得到季度的数据，1 年 12 个月的数字相加可以得到年度的数据。

由于事实表一般记录数都非常多，而且会随着时间数据越积累越多，用户直接在上面做汇总计算来观察一些度量值的时候，可能需要等很长时间才能得到汇总计算的结果。所以在确定了数据的粒度后，为了提高数据仓库的使用性能，可以根据用户的要求，可以按照维度的不同组合来设计聚合模型，从而在事实表的基础上再设置一些聚合表，存储一些在事实表的基础上预先汇总好的聚合数据，让用户获得更好的查询性能。

3．分割

分割是数据仓库中的数据存储中的另外一个重要概念，它的目的在于提高效率。它是将数据分散到各自的物理单元中去，以便能分别独立处理，实现查询操作的并行。有许多数据分割的标准可供参考，如时间、地域、业务领域等，也可以是其组合。一般而言，分割标准总应包括一些能让它十分自然而且分割均匀的项目，如时间项等。

6.2.7　元数据

往往一个数据仓库需要包容和整合成千上万的信息内容，内容的多样性使数据仓库的数据结构显得异常的庞大。因此，要简单地用一种不需要言传的方式来描述一个数据仓库的内容和结构是不可能的事情，因而在从开发到运行维护的整个数据仓库生命周期中，如何描述数据仓库里面有的东西成了一件非常重要的事情。

元数据（Meta-data）通常定义为“关于数据的数据”，是描述和管理数据仓库自身

内容对象，用来表示数据项的意义及其在系统各组成部件之间的关系的数据。实际上，数据仓库所提供的“统一的企业级的信息视图”能力，主要就是靠元数据来体现，如果把建设数据仓库比喻成搭建房子，元数据就是房子的“图纸”。是从广义上来讲，用元数据来描述数据仓库对象的任何东西——无论是一个表、一个列、一个查询、一个商业规则，或是数据仓库内部的数据转移。它在数据源的抽取、数据加工、访问与使用等过程中都会存在。实现元数据管理的主要目标就是使企业内部元数据的定义标准化。数据仓库的维护工具可以根据元数据的“指示”完成数据的抽取、清洗和转换，并做适度的汇总。数据仓库的元数据包括以下内容。

（1）数据资源：包括各个数据源的模型，描述源数据表字段属性及业务含义，源数据到数据仓库的映射关系。

（2）数据组织：数据仓库、数据集市表的结构、属性及业务含义，多维结构等。

（3）数据应用：查询与报表输出格式描述、OLAP、数据挖掘等的数据模型的信息展现、商业术语。

（4）数据管理：这里包括数据仓库过程以及数据仓库操作结果的模型，包括描述数据抽取和清洗规则、数据加载控制、临时表结构、用途和使用情况、数据汇总控制。

元数据贯穿整个数据仓库项目，所有数据处理环节必须最大化的参照元数据，这样才能保证数据仓库项目不会因为不断增长的数据多样性而失去秩序，特别是现行应用的异构性与分布性越来越普遍的情况下，统一的元数据就愈发重要了。“信息孤岛”曾经是很多企业对其应用现状的一种抱怨和概括，而合理的元数据则会有效的描绘出信息的关联性，从而大大降低了数据仓库后期的维护和运行成本。

按照元数据的使用情况和面向对象的不同，可以将元数据分为业务元数据、技术元数据、操作元数据。

1．业务元数据

业务元数据用业务名称、定义、描述和别名来表示数据仓库和业务系统中的各种属性，直接供最终用户使用。业务元数据使最终用户能够更好理解、使用数据仓库，成为最终用户在数据仓库中的业务信息地图。

业务元数据在系统的数据仓库中的体现是全方位的，例如，最终用户通过浏览元数据可以清晰地了解当前指标代表什么业务、如何计算得出的、以什么为单位等相关描述信息。

2．技术元数据

技术元数据描述了源系统、数据转换、抽取过程、工作流、加载策略以及目标数据库的定义等。技术元数据可供信息系统人员和一部分最终用户使用，用来进行影响分析、变化管理、数据库优化、任务调度和安全管理等。

OLTP 业务系统和数据仓库 OLAP 分析系统之间存在复杂、多方面的区别，因此，数据在业务系统和分析系统之间的处理、加载也是复杂和涉及多方面的。技术元数据对

数据在两个系统间处理、加载的规则、过程、相关策略进行了描述。

3．操作元数据

操作元数据描述了目标表中的信息，如粒度、创建目标表和索引的信息、刷新时间、记录数、按时执行任务的设置以及有权访问数据的用户。操作元数据用于数据仓库的维护和分布。

虽然元数据依据具体应用特点分为业务元数据、技术元数据和操作元数据，但是，在实际应用中以上三类元数据是相互参照和关联的。只有业务、技术、操作之间的协调和互补才能充分发挥数据仓库的潜能，提高数据仓库的利用效率。

4．元数据标准

元数据标准（Common Warehouse Metamodel，CWM）定义一个描述数据源、数据目的、转换、分析的元数据框架，以及定义建立和管理数据仓库的过程和操作，提供使用信息的继承。

CWM 主要基于以下三个工业标准：

（1）统一建模语言（Unified Modeling Language，UML），是用来对软件密集系统进行可视化建模的一种语言。UML 为面向对象开发系统的产品进行说明、可视化和编制文档的一种标准语言。有关 UML 的更加详细的知识，将在第 13 章进行介绍。

（2）元对象机制（Meta Object Facility，MOF），这是元模型和模型库的标准，提供在异构环境下的数据交换的接口。MOF 是模型驱动架构（Model Driven Architecture，MDA）的核心。

（3）元数据交换（XML Metadata Interchange，XMI），这是一个用于元数据交换的一个标准。XMI 的目的在于帮助使用 UML 以及不同语言和开发工具的程序员彼此交换数据模型。

6.3　数据仓库设计与开发

在 6.2 节中，详细介绍了与数据仓库的相关概念，以及数据仓库的特点和与操作型数据的区别，那么，在实际应用中，究竟如何设计与开发一个数据仓库系统呢？这就是本节要介绍的内容。

6.3.1　数据仓库的设计过程

数据仓库是以商业智能应用为目的而设计的，所以从广义的数据仓库设计应该包括数据仓库中的数据模型设计和在其上面的数据分析应用设计两个方面，数据仓库的应用和数据仓库的设计一脉相承，共同构成整个数据仓库系统的生命周期，这个周期包括三个阶段，分别是规划阶段、设计实施阶段和使用维护阶段，如图 6-19 所示。

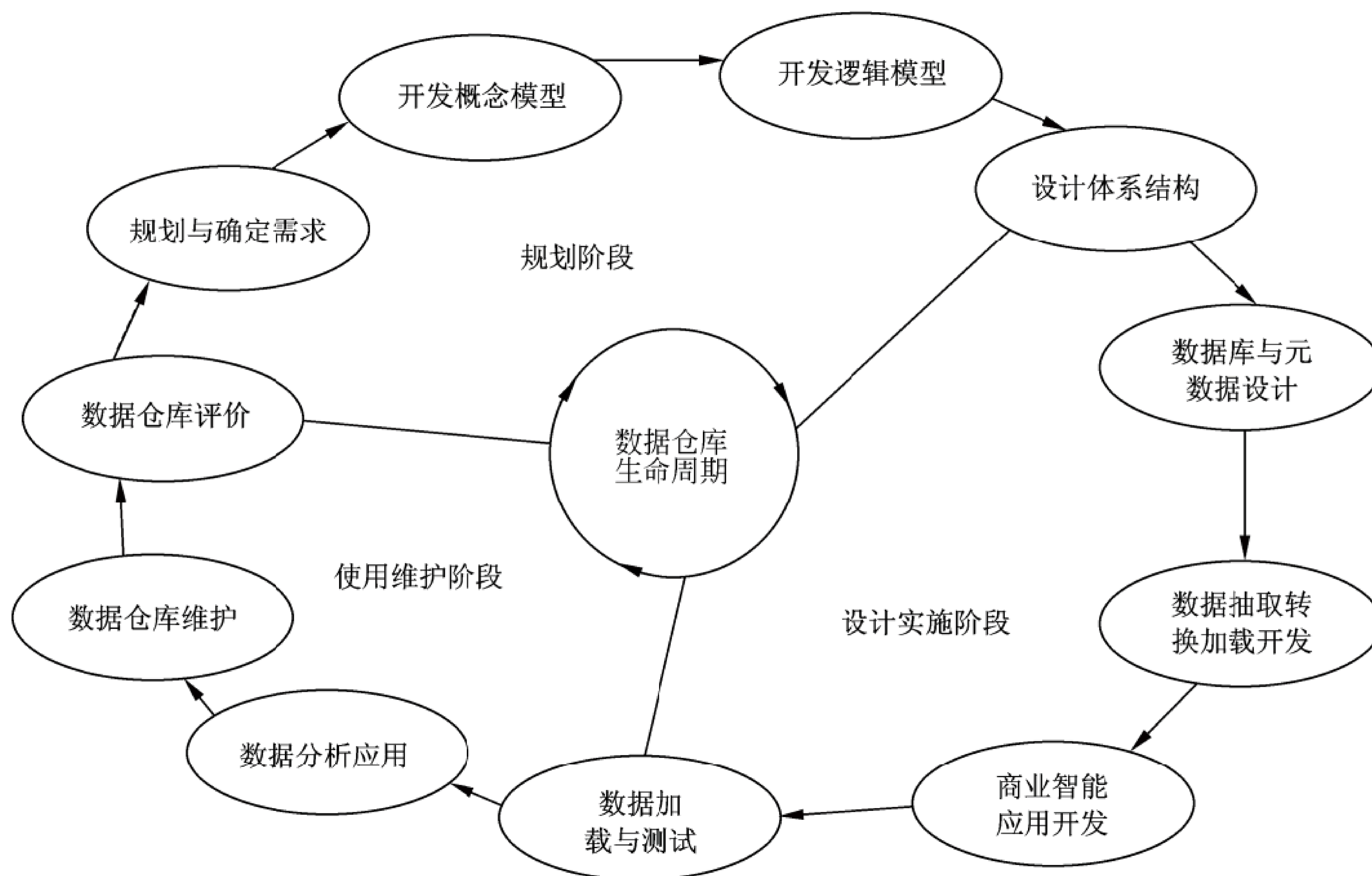


图 6-19 数据仓库的生命周期

正如 6.1.2 节描述的商业智能是一个过程，这三个阶段也构成了一个不断循环、完善和提高的过程，一般情况下数据仓库不可能在一个循环过程中完成，而是需要多次循环开发，每次循环会为系统增加新的功能，使数据仓库的应用得到新的提高。

微观上的数据仓库设计实际上是指数据仓库的数据模型设计，这个层面的主要任务是数据建模，确定数据仓库中数据的内容及其构成关系，如图 6-20 所示。和数据库的设计一样，数据仓库的设计也是在概念模型、逻辑模型和物理模型的依次转换过程中实现，而作为建设数据仓库的“图纸”，元数据模型则自始至终伴随着数据仓库的开发、实施和使用。

6.3.2 创建数据仓库的方式

总体来说，创建数据仓库的方式主要有三种，分别是自上而下、自下而上和元数据驱动的实施方法。本节就简单地介绍这三种方法。

1．自上而下

把数据仓库定义为一个大系统，“全局考虑，全面实施”，建立适合企业信息共性需求的完整的数据模型，然后从业务运营系统中提取数据，进行数据的清洗、合并、规范化和合理化，并加载到数据仓库中，形成企业统一的数据集成平台，最后可以根据部门个性需要将数据仓库的数据分发到面向主题的数据集市中。

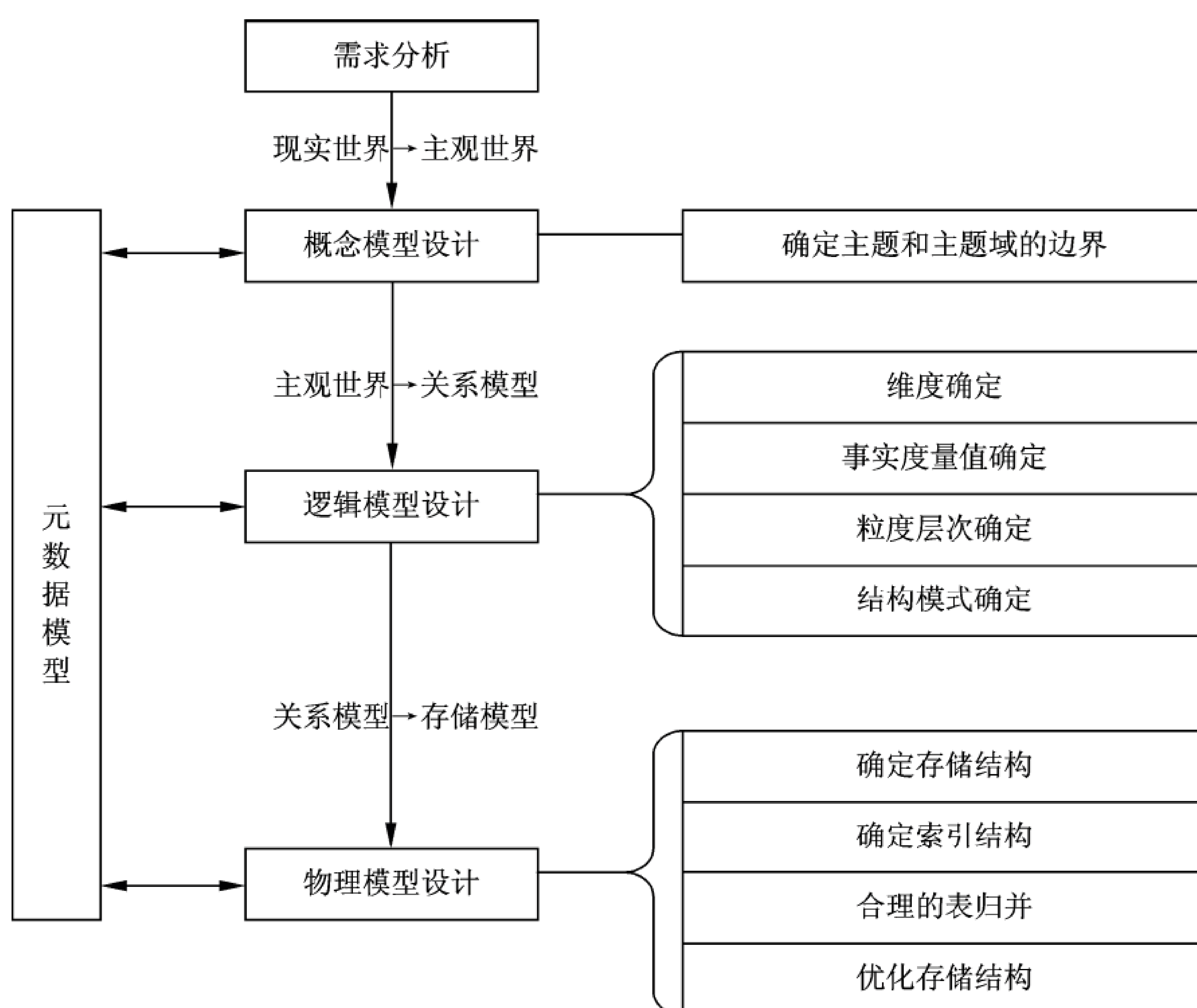


图 6-20　数据仓库数据模型设计的步骤

自上而下开发方法的优点如下。

（1）企业统一的数据集成平台。

（2）集中化的控制管理。

（3）数据容易分发到各个数据集市中。

自上而下开发方法的缺点如下。

（1）开发过程复杂，费用高。

（2）开发时间长，难以满足快速变化的业务需求。

（3）需要进行大量的业务需求分析，需要大量的资源。

（4）结构比较僵化，比较难以扩展。

2．自下而上

大量的旧系统，要想在短时间内进行数据的合理性和完整性统一是相当困难的，而市场变化和企业决策规则变化不允许花大量的时间和精力去建立一个满足日后需求，但不满足现在变化的系统。自下而上的开发方法就是根据特定的业务主题，“分部门考虑，分部门实施”，可以在很短的时间内实现部门级的数据集市，多个数据集市组成企业联邦制的数据仓库。

自下而上开发方法的优点如下。

（1）可以并行开发。

（2）见效快。

（3）分散化的资源和管理控制。

自下而上开发方法的缺点如下。

（1）很难协调各个数据集市的建设。

（2）可能存在着部门之间的政治斗争和数据集市归属问题。

（3）如果采用不同的技术建立起来的数据集市，最终造成多个相互独立、互不兼容的“烟囱式”数据集市，给维护和数据共享带来很大的障碍。

（4）多种数据源采集系统，可能造成对业务系统的冲击和数据的不一致。

3．元数据驱动的实施方法

元数据驱动、螺旋上升的数据仓库建立的过程就是“建立元数据——构造数据仓库/集市”的不断循环、不断上升的过程，如图 6-21 所示。

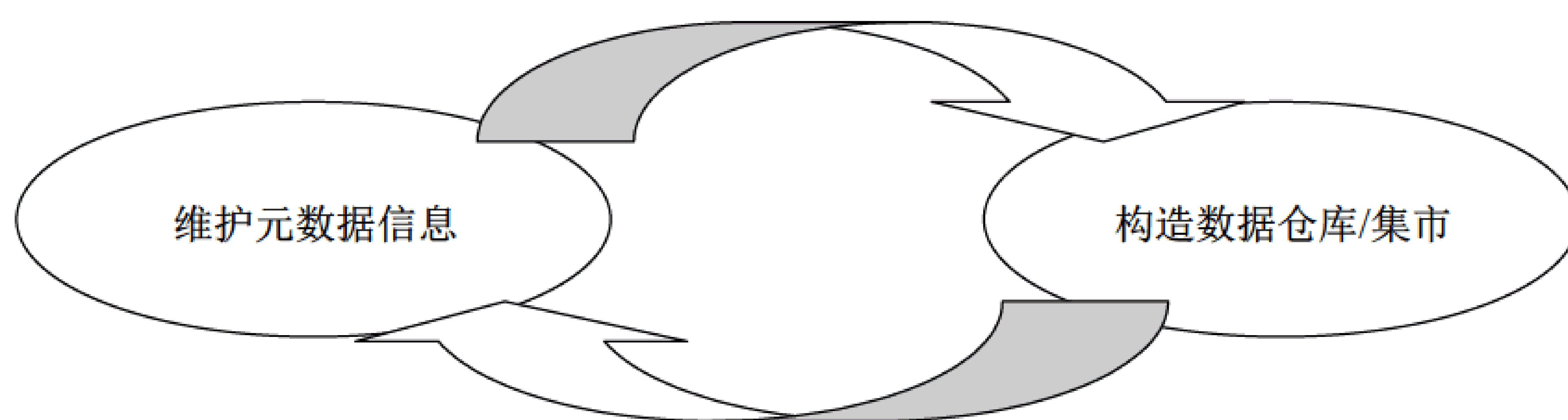


图 6-21　不断循环、螺旋上升的元数据驱动方法

元数据驱动的数据仓库开发过程如图 6-22 所示，可以细分为以下阶段。

1）建立元数据

（1）根据业务数据源，外部数据源定义元数据。

（2）添加和更新维护元数据的内容和属性。

（3）定义元数据使用规则。

（4）声明元数据联合使用的规则。

2）构造数据仓库/集市

（1）基于元数据进行数据抽取/清洗/转换/聚合/加载/分布。

（2）基于元数据进行前端应用界面定制。

从另外一个层面来说，第 1 步中构造和维护元数据信息可以看作虚拟地构造商业智能平台的过程，能否在第 2 步实现这种虚拟的构造过程和实际的构造过程的有机结合，是落实元数据驱动的关键。

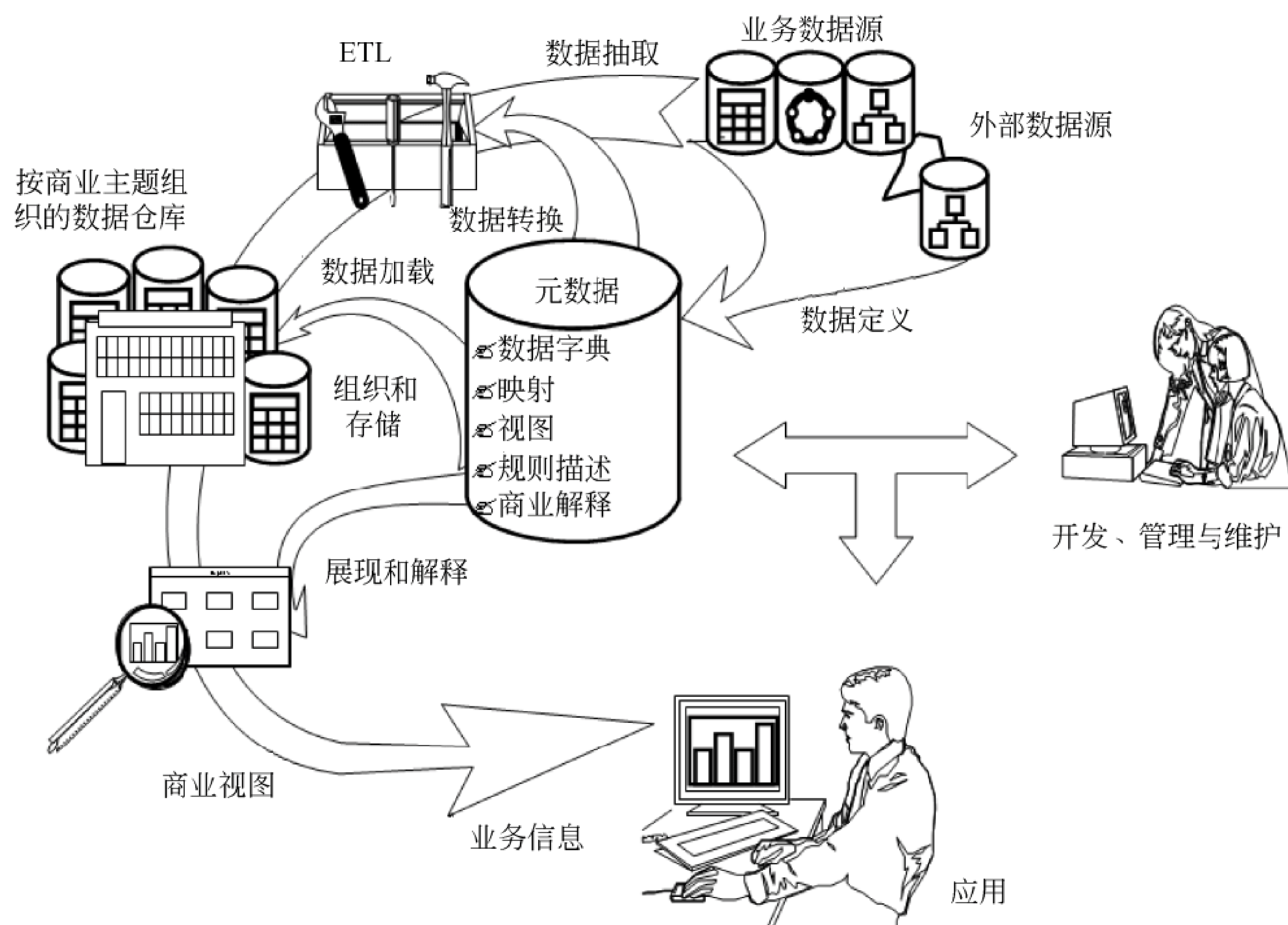


图 6-22　以元数据为中心的开发方法

这种开发方式优点是显而易见的，包括：

（1）建立企业数据的统一视图。

（2）有统一的元数据管理。

（3）具有灵活可扩展的体系结构。

（4）分步式开发，螺旋式上升，既能快速看到效果，又保证系统的连续性、一致性。

这种开发方式的主要缺点就是如何真正实现这种方式，提取和维护元数据并不是一件很难的事情。而在实际的实施过程中，如何真正地实现元数据的驱动则不是一件容易的事情，由于传统程序开发思想的影响，很多开发者对需求问题的解决更多地依赖于程序设计，这样使得很多控制逻辑很难被抽象出来，因此也难于把数据的处理过程标准化、规范化，这种以程序驱动的方式很容易把所谓的元数据驱动流于形式。

本章参考文献

[1] 李艳. 商业智能是一种解决方案. 中国计算机用户，2003

[2] 九城集团 BI 事业部. 九城信息智能平台白皮书. 2002

[3] Kevin Strange. Data Warehouse Data Model: Neutrality is the Key. Gartner Article

MRR-0798-17，1998

[4] 陈武，袁国忠. 企业数据仓库规划、建立与实现. 北京：人民邮电出版社，2000

[5] 佚名. OLAP 技术应用研究. http://www.dwway.com/document.php?id=682

[6] 朱德利. SQL Server 2005 数据挖掘与商业智能完全解决方案. 北京：电子工业出版社，2007

[7] IBM 公司. IBM 商业智能解决方案综述. 2004

[8] 郑晓舟，胡睿. 商务智能：信息知识利润. 北京：电子工业出版社，2005

[9] 王志海，林友芳. 建设数据仓库. 北京：机械工业出版社，2003

[10] 张峰岭. 用需求来创造价值：论数据仓库与商业智能需求与需求分析. 2006 中国软件工程大会会刊，2006

第 7 章　数 据 挖 掘

数据挖掘也是商业智能的一种应用，商业智能的应用范围可以分为三个层次，分别是报表、分析和挖掘，其中报表是解决“现在是怎样的问题”，分析是解决“为什么会有这样的问题”，而挖掘是解决“以后会有怎样的问题”。简单地说，数据挖掘就是从企业的数据库的大量数据中，抽取出潜在的、有价值的信息（业务模型或业务规则）的过程。

本章首先描述数据挖掘的概念，然后介绍目前常见的数据挖掘技术，再介绍一下数据挖掘系统的结构和数据挖掘的过程，最后介绍一些数据挖掘的热点应用。本章的目的是让读者了解数据挖掘的基本知识，由于数据挖掘技术的实现涉及到比较深的数学统计、人工智能和神经网络等领域的理论，而实际的应用则要结合具体的工具和数据场景，这些依赖于读者去阅读数据挖掘方面的专著或具体工具的产品描述。

7.1　数据挖掘概述

数据挖掘使数据库技术进入了一个更高级的阶段，它不仅能对过去的数据进行查询和遍历，并且能够找出过去数据之间的潜在联系，从而促进信息的传递。现在数据挖掘技术在商业应用中已经可以马上投入使用，因为对这种技术进行支持的三种基础技术已经发展成熟，他们是海量数据搜集、强大的多处理器计算机和数据挖掘算法。

7.1.1　数据挖掘的定义

从技术上来看，数据挖掘就是从大量的、不完全的、有噪声的、模糊的、随机的实际应用数据中，提取隐含在其中的、人们事先不知道的，但又是潜在有用的信息和知识的过程。这个定义包括好几层含义：

（1）数据源必须是真实的、大量的，难免也是含噪声的。

（2）发现的是用户感兴趣的知识。

（3）发现的知识要可接受、可理解、可运用。

（4）并不要求发现放之四海皆准的知识，仅支持特定的发现问题。

还有很多和这一术语相近似的术语，如从数据库中发现知识（Knowledge Discovery in Databases，KDD）、数据分析、数据融合（Data Fusion）以及决策支持等。

从广义上理解，数据、信息也是知识的表现形式，但是人们更把概念、规则、模式、规律和约束等看作知识。原始数据可以是结构化的，如关系数据库中的数据；也可以是半结构化的，如文本、图形和图像数据；甚至是分布在网络上的异构型数据。发现知识

的方法可以是数学的，也可以是非数学的；可以是演绎的，也可以是归纳的。发现的知识可以被用于信息管理，查询优化，决策支持和过程控制等，还可以用于数据自身的维护。因此，数据挖掘是一门交叉学科，它把人们对数据的应用从低层次的简单查询，提升到从数据中挖掘知识，提供决策支持。在这种需求牵引下，汇聚了不同领域的研究者，尤其是数据库技术、人工智能技术、数理统计、可视化技术、并行计算等方面的学者和工程技术人员，投身到数据挖掘这一新兴的研究领域，形成新的技术热点。

从商业角度来看，数据挖掘是一种新的商业信息处理技术，其主要特点是对商业数据库中的大量业务数据进行抽取、转换、分析和其他模型化处理，从中提取辅助商业决策的关键性数据。

简而言之，数据挖掘其实是一类深层次的数据分析方法。数据分析本身已经有很多年的历史，只不过在过去数据收集和分析的目的是用于科学研究，另外，由于当时计算能力的限制，对大数据量进行分析的复杂数据分析方法受到很大限制。现在，由于各行业业务自动化的实现，商业领域产生了大量的业务数据，这些数据不再是为了分析的目的而收集的，而是由于纯机会的（opportunistic）商业运作而产生。分析这些数据也不再是单纯为了研究的需要，更主要是为商业决策提供真正有价值的信息，进而获得利润。但所有企业面临的一个共同问题是：企业数据量非常大，而其中真正有价值的信息却很少，因此从大量的数据中经过深层分析，获得有利于商业运作、提高竞争力的信息，就像从矿石中淘金一样，数据挖掘也因此而得名。

因此，数据挖掘可以描述为：按企业既定业务目标，对大量的企业数据进行探索和分析，揭示隐藏的、未知的或验证已知的规律性，并进一步将其模型化的先进有效的方法。

数据挖掘与传统的数据分析（如查询、报表、联机应用分析）的本质区别是数据挖掘是在没有明确假设的前提下去挖掘信息、发现知识。数据挖掘所得到的信息应具有先未知、有效和可实用三个特征。先前未知的信息是指该信息是预先未曾预料到的，既数据挖掘是要发现那些不能靠直觉发现的信息或知识，甚至是违背直觉的信息或知识，挖掘出的信息越是出乎意料，就可能越有价值。在商业应用中最典型的例子就是一家连锁店通过数据挖掘发现了小孩尿布和啤酒之间有着惊人的联系。

希赛教育专家提示：数据挖掘技术从一开始就是面向应用的。它不仅是面向特定数据库的简单检索查询调用，而且要对这些数据进行微观、中观乃至宏观的统计、分析、综合和推理，以指导实际问题的求解，企图发现事件间的相互关联，甚至利用已有的数据对未来的活动进行预测。例如，加拿大 BC 省电话公司要求加拿大 SimonFraser 大学 KDD 研究组，根据其拥有十多年的客户数据，总结、分析并提出新的电话收费和管理办法，制定既有利于公司又有利于客户的优惠政策。这样一来，就把人们对数据的应用，从低层次的末端查询操作，提高到为各级经营决策者提供决策支持。这种需求驱动力，比数据库查询更为强大。

7.1.2　数据挖掘的功能

数据挖掘通过预测未来趋势及行为，做出前摄的、基于知识的决策。数据挖掘的目标是从数据库中发现隐含的、有意义的知识，主要有以下 5 类功能。

（1）自动预测趋势和行为。数据挖掘自动在大型数据库中寻找预测性信息，以往需要进行大量手工分析的问题如今可以迅速直接由数据本身得出结论。一个典型的例子是市场预测问题，数据挖掘使用过去有关促销的数据来寻找未来投资中回报最大的用户，其他可预测的问题包括预报破产以及认定对指定事件最可能做出反应的群体。

（2）关联分析。数据关联是数据库中存在的一类重要的可被发现的知识。若两个或多个变量的取值之间存在某种规律性，就称为关联。关联可分为简单关联、时序关联、因果关联。关联分析的目的是找出数据库中隐藏的关联网。有时并不知道数据库中数据的关联函数，即使知道也是不确定的，因此关联分析生成的规则带有可信度。

（3）聚类。数据库中的记录可被划分为一系列有意义的子集，即聚类。聚类增强了人们对客观现实的认识，是概念描述和偏差分析的先决条件。聚类技术主要包括传统的模式识别方法和数学分类学。20 世纪 80 年代初，Mchalski 提出了概念聚类技术及其要点是，在划分对象时不仅考虑对象之间的距离，还要求划分出的类具有某种内涵描述，从而避免了传统技术的某些片面性。

（4）概念描述。概念描述就是对某类对象的内涵进行描述，并概括这类对象的有关特征。概念描述分为特征性描述和区别性描述，前者描述某类对象的共同特征，后者描述不同类对象之间的区别。生成一个类的特征性描述只涉及该类对象中所有对象的共性。生成区别性描述的方法很多，如决策树方法、遗传算法等。

（5）偏差检测。数据库中的数据常有一些异常记录，从数据库中检测这些偏差很有意义。偏差包括很多潜在的知识，如分类中的反常实例、不满足规则的特例、观测结果与模型预测值的偏差、量值随时间的变化等。偏差检测的基本方法是，寻找观测结果与参照值之间有意义的差别。

7.2　数据挖掘常用技术

知识发现中的关键技术是进行模式和关系识别的算法，下面介绍几种数据挖掘技术，它们分别从不同的角度进行数据挖掘和知识发现。

1．决策树

利用信息论中的互信息（信息增益）寻找数据库中具有最大信息量的字段，建立决策树的一个节点，再根据字段的不同取值建立树的分支，在每个分支子集中重复建树的下层节点和分支的过程，即可建立决策树。国际上最有影响和最早的决策树方法是 Quiulan 研制的 ID3 方法，它对越大的数据库效果越好。在 ID3 方法的基础上，又演化

为能处理连续属性的 C4.5。

决策树构造的输入是一组带有类别标记的例子，构造的结果是一棵二叉或多叉树。二叉树的内部节点（非叶子节点）一般表示为一个逻辑判断，如形式为（$ai = vi$）的逻辑判断，其中 ai 是属性，vi 是该属性的某个属性值；树的边是逻辑判断的分支结果。多叉树的内部节点是属性，边是该属性的所有取值，有几个属性值，就有几条边。树的叶子节点都是类别标记。构造决策树的方法是采用自上而下的递归构造。以多叉树为例，它的构造思路是，如果训练例子集合中的所有例子是同类的，则将之作为叶子节点，节点内容即是该类别标记。否则，根据某种策略选择一个属性，按照属性的各个取值，把例子集合划分为若干子集合，使得每个子集上的所有例子在该属性上具有同样的属性值。然后再依次递归处理各个子集。这种思路实际上就是“分而治之”（divide-and-conquer）的道理。二叉树同理，差别仅在于要选择一个好的逻辑判断。

2．分类

分类在数据挖掘中是一项非常重要的任务。该算法将数据按含义划分成组，可用此算法生成感兴趣的侧面，可用于自动发现类，如模式识别、侧面生成、线性聚簇和概念聚簇等。分类的目的是学会一个分类函数或分类模型（也称为分类器），该模型能把数据库中的数据项映射到给定类别中的某一个。分类和回归都可用于预测。预测的目的是从利用历史数据纪录中自动推导出对给定数据的推广描述，从而能对未来数据进行预测。和回归方法不同的是，分类的输出是离散的类别值，而回归的输出则是连续数值。

要构造分类器，需要有一个训练样本数据集作为输入。训练集由一组数据库记录或元组构成，每个元组是一个由有关字段（又称属性或特征）值组成的特征向量，除了这些外，训练样本还有一个类别标记。一个具体样本的形式可为：（$v1$，$v2$，...，vn；c）；其中 vi 表示字段值，c 表示类别。

分类器的构造方法有统计方法、机器学习方法、神经网络方法等等。统计方法包括贝叶斯法和非参数法（近邻学习或基于事例的学习），对应的知识表示则为判别函数和原型事例。机器学习方法包括决策树法和规则归纳法，前者对应的表示为决策树或判别树，后者则有两种，分别是决策表（decision list）和（平行）产生式规则。神经网络方法主要是反向传播（Back Propagation，BP）算法，它的模型表示是前向反馈神经网络模型（由代表神经元的节点和代表联接权值的边组成的一种体系结构），BP 算法本质上是一种非线性判别函数。

3．粗糙集

粗糙集（Rough Set）的研究主要基于分类，分类和概念（concept）同义，一种类别对应于一个概念(类别一般表示为外延即集合，而概念常以内涵的形式表示如规则描述)。知识由概念组成，如果某知识中含有不精确概念，则该知识不精确。粗糙集对不精确概念的描述方法是：通过上近似概念和下近似概念这两个精确概念来表示。一个概念（或集合）的下近似（lower approximation）概念（或集合）指的是，其下近似中的元素肯定

属于该概念；一个概念（或集合）的上近似（upper approximation）概念（或集合）指的是，其上近似中的元素可能属于该概念。在数据库中，将行元素看成对象，列元素看成属性（分为条件属性和决策属性）。等价关系 R 定义为不同对象在某个（或几个）属性上取值相同，这些满足等价关系的对象组成的集合称为该等价关系 R 的等价类。条件属性上的等价类 E 与决策属性上的等价类 Y 之间有三种情况。

（1）下近似：Y 包含 E。

（2）上近似：Y 和 E 的交非空。

（3）无关：Y 和 E 的交为空。

对下近似建立确定性规则，对上近似建立不确定性规则（含可信度），对无关情况不存在规则。

粗糙集方法为 KDD 提供了一种新的方法和工具。首先，KDD 研究的实施对象多为关系型数据库。关系表可被看作为粗糙集理论中的决策表，这给粗糙集方法的应用带来极大的方便。第二，现实世界中的规则有确定性的，也有不确定性的。从数据库中发现不确定性的知识，为粗糙集方法提供了用武之地。第三，从数据中发现异常，排除知识发现过程中的噪声干扰也是粗糙集方法的特长。第四，运用粗糙集方法得到的知识发现算法有利于并行执行，这可极大地提高发现效率。对于大规模数据库中的知识发现来说，这正是求之不得的。第五，KDD 中采用的其他技术，如神经网络方法，不能自动地选择合适的属性集，而利用粗糙集方法进行预处理，去掉多余属性，可提高发现效率，降低错误率。第六，粗糙集方法比模糊集方法或神经网络方法在得到的决策规则和推理过程方面更易于被证实和检测。

4．神经网络

神经网络通过学习待分析数据中的模式来构造模型，它可对隐式类型进行分析，适用于模型化非线性的、复杂的或高噪声的数据。它模拟人脑神经元结构，由“神经元”互联，或按层组织的节点构成。通常，神经模型由三个层次组成：输入、中间和输出层。每个神经元求得输入值，再计算总输入值，由过滤机制（如阈值）比较总输入，然后确定它自己的输出值。可通过连接一组神经元来模型化复杂行为。当修改连接层得“连接度”或参数时，神经网络就进行了学习或“训练”。神经网络的知识体现在网络连接的权值上，是一个分布式矩阵结构；神经网络的学习体现在神经网络权值的逐步计算上（包括反复迭代或累加计算）。以 Hebb 学习规则为基础，可以建立三大类多种神经网络模型。

（1）前馈式网络：它以感知机、反向传播模型、函数型网络为代表，可用于预测、模式识别等方面。

（2）反馈式网络：它以 Hopfield 的离散模型和连续模型为代表，分别用于联想记忆和优化计算。

（3）自组织网络：它以 ART 模型、Koholon 模型为代表，用于聚类。

神经网络可按管理模式或非管理模式来学习，在管理模式中，神经网络要预测现有

示例可能带来得结果，它将预测结果与目标答案相比较并从错误中进行学习。管理模式得神经网络可用于预测、分类和时间序列模型。非管理模式得学习在描述数据时很有效，但却不用于预测结果。非管理模式的神经网络创建自己的类描述、合法性验证和操作，它与数据模式无关。

5．关联规则

关联规则是形式如下的一种规则：在购买面包和黄油的顾客中，有 90%的人同时也买了牛奶（面包+黄油=>牛奶）。用于关联规则发现的主要对象是事务型数据，其中针对的应用则是售货数据，也称货篮数据。一个事务一般由如下几个部分组成：事务处理时间、一组顾客购买的物品，有时也有顾客标识号。关联规则就是指搜索业务系统中的所有细节和事务，从中找出重复出现概率很高的模式，它以大的事务数据库为基础，其中每个事务都被定义为一系列相关数据项。用关联找出所有能把一组事件或数据项与另一套事件或数据项联系起来的规则。对关系数据集可以使用这种处理，此类数据是用标准 SQL 谓词逻辑定义的。关联算法的目的是成为 SQL 的扩充，这样这种算法就可以通过规范的查询技术应用于受限的关系数据集。这些算法必须有高度的适应性和动态性。为了找到关系模式，要查看的数据集会有所变化，关联发生的最小百分比规则会发生变化。

6．概念树

对数据库中记录的属性字段按归类方式进行抽象，建立起来的层次结构称为概念树。以软考为例，“型号”概念树的最下层是具体考试级别（如网络工程师、系统分析师等），它的直接上层是资格小类（如中级资格、高级资格等），资格小类的直接上层是专业大类（如计算机网络、计算机软件等），再上层就是整个软考。利用概念树提升的方法可以大大浓缩数据库中的记录。对多个属性字段的概念树进行提升，将得到高度概括的知识基表，然后再将它转换成规则。

7．遗传算法

遗传算法是模拟生物进化过程的算法，由三个基本算子组成。

（1）繁殖（选择）：从一个旧种群（父代）选出生命力强的个体，产生新种群（后代）的过程。

（2）交叉（重组）：选择两个不同个体（染色体）的部分（基因）进行交换，形成新个体。

（3）变异（突变）：对某些个体的某些基因进行变异（1 变 0，0 变 1）。

这种遗传算法可起到产生优良后代的作用。这些后代需满足适应值，经过若干代的遗传，将得到满足要求的后代（问题的解）。遗传算法已在优化计算和分类机器学习方面发挥了显著作用。

8．依赖性分析

依赖性分析在数据仓库的条目或对象间抽取依赖性，它展示了数据间未知的依赖关系，并有可能描述成关注性数据项间的因果关系，可以用该分析方法从某一数据对象的

信息来推断另一数据对象的信息，依赖性是一个带有置信度因子的可能值。

9．公式发现

在工程和科学数据库（由实验数据组成）中，对若干数据项（变量）进行一定的数学运算，求得相应的数学公式。例如，比较典型的 BACON 发现系统完成了对物理学中大量定律的重新发现。其基本思想是，对数据项进行初等数学运算（加、减、乘、除等），形成组合数据项，若它的值为常数项，就得到了组合数据项等于常数的公式。

10．统计分析方法

在数据库字段项之间存在两种关系，一是函数关系（能用函数公式表示的确定性关系），二是相关关系（不能用函数公式表示，但仍是相关确定关系）。对它们的分析可以采用回归分析、相关分析或主成分分析。

11．模糊论方法

利用模糊集合理论对实际问题进行模糊评判、模糊决策、模糊模式识别和模糊聚类分析。模糊性是客观存在的。系统的复杂性越高，精确化能力就越低，即模糊性越强。这是 Zadeh 总结出的互克性原理。

12．可视化技术

可视化分析可给出带有多变量的图形化分析数据，帮助分析员进行发现，它可使分析员同时显示多个变量间的关系。可视化数据分析技术拓宽了传统的图表功能，使用户对数据的剖析更清楚。例如，把数据库中的多维数据变成多种图形，这对揭示数据的状况、内在本质及规律性起了很大作用。

7.3　数据挖掘的结构与流程

数据挖掘和知识发现系统用于发现先前未知的知识，知识是一种描述规律的信息，表现为数据元素间的关系或模式，这些数据与特定的领域和任务相关，并且是令人感兴趣的和有用的。

7.3.1　数据挖掘系统的结构

数据挖掘系统的逻辑视图如图 7-1 所示。

（1）知识发现系统管理器。控制并管理知识发现过程，分析员录入和知识库中的信息用于驱动数据选择过程、抽取算法选择及使用过程和发现评价过程三个过程。

（2）知识库和分析员录入。知识库包含源于多方面的必须的信息，分析人员可以将元数据输入数据仓库中来描述数据仓库的数据结构，输入关键数据字段、规则、数据层次等。

（3）数据仓库的数据访问接口。知识发现系统利用数据库的查询机制从数据仓库中提取数据，可使用 SQL 查询语言，结合知识库中的数据仓库元数据指导从数据仓库中提

取需要的数据。

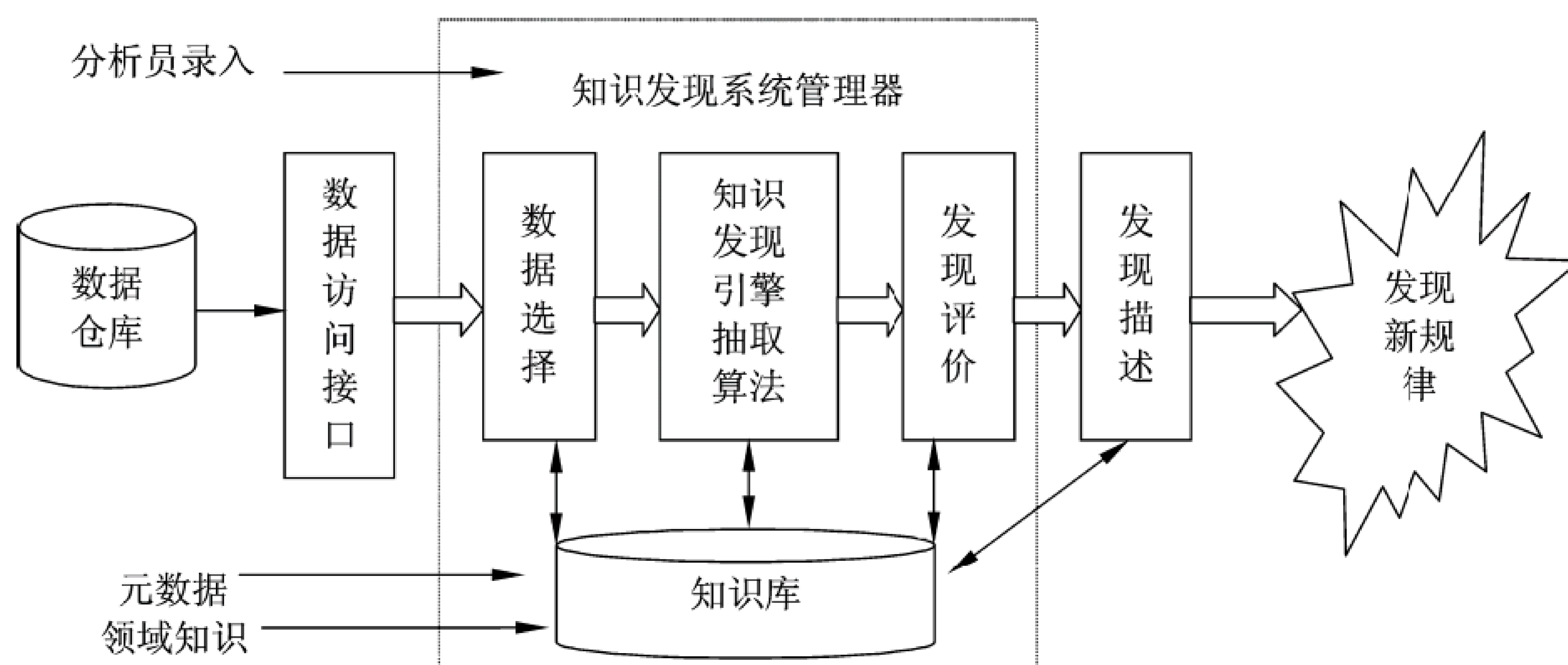


图 7-1 数据挖掘系统的逻辑结构图

（4）数据选择。确定从数据仓库需要抽取的数据及数据结构。知识库指导选取要抽取的数据及抽取方式。

（5）知识发现引擎。将知识库中的抽取算法提供给数据选取抽取的数据，目的是要抽取数据元素间的模式和关系。

（6）发现评价。分析人员要寻找关注性的数据模式，数据仓库潜在地具有宿主模式，选出那些关注性信息。

（7）发现描述。提供两种功能，一种是以发现评价辅助分析员在知识库中保存所发现的关注性结果以备将来引用和使用；另一种是保持发现与决策者的通信。

7.3.2 数据挖掘的流程

数据挖掘的流程大致如下：

（1）问题定义。在开始数据挖掘之前最先的也是最重要的要求就是熟悉背景知识，弄清用户的需求。缺少了背景知识，就不能明确定义要解决的问题，就不能为挖掘准备优质的数据，也很难正确的解释得到的结果。要想充分发挥数据挖掘的价值，必须对目标要有一个清晰明确的定义，即决定到底想干什么。

（2）建立数据挖掘库。要进行数据挖掘必须收集要挖掘的数据资源。一般建议把要挖掘的数据都收集到一个数据库中，而不是采用原有的数据库或数据仓库。这是因为大部分情况下需要修改要挖掘的数据，而且还会遇到采用外部数据的情况；另外，数据挖掘还要对数据进行各种纷繁复杂的统计分析，而数据仓库可能不支持这些数据结构。

（3）分析数据。分析数据就是通常所进行的对数据深入调查的过程。从数据集中找出规律和趋势，用聚类分析区分类别，最终要达到的目的就是搞清楚多因素相互影响的、十分复杂的关系，发现因素之间的相关性。

（4）调整数据。通过上述步骤的操作，对数据的状态和趋势有了进一步的了解，这时要尽可能对问题解决的要求能进一步明确化、进一步量化。针对问题的需求对数据进行增删，按照对整个数据挖掘过程的新认识组合或生成一个新的变量，以体现对状态的有效描述。

（5）模型化。在问题进一步明确，数据结构和内容进一步调整的基础上，就可以建立形成知识的模型。这一步是数据挖掘的核心环节，一般运用神经网络、决策树、数理统计、时间序列分析等方法来建立模型。

（6）评价和解释。上面得到的模式模型，有可能是没有实际意义或没有实用价值的，也有可能是其不能准确反映数据的真实意义，甚至在某些情况下是与事实相反的，因此需要评估，确定哪些是有效的、有用的模式。评估的一种办法是直接使用原先建立的挖掘数据库中的数据来进行检验，另一种办法是另找一批数据并对其进行检验，再一种办法是在实际运行的环境中取出新鲜数据进行检验。

数据挖掘过程的分步实现，不同的步骤会需要有不同专长的人员，他们大体可以分为三类。

（1）业务分析人员：要求精通业务，能够解释业务对象，并根据各业务对象确定出用于数据定义和挖掘算法的业务需求。

（2）数据分析人员：精通数据分析技术，并对统计学有较熟练的掌握，有能力把业务需求转化为数据挖掘的各步操作，并为每步操作选择合适的技术。

（3）数据管理人员：精通数据管理技术，并从数据库或数据仓库中收集数据。

希赛教育专家提示：数据挖掘是一个多种专家合作的过程，也是一个在资金上和技术上高投入的过程。这一过程要反复进行，在反复过程中，不断地趋近事物的本质，不断地优选问题的解决方案。

7.4　数据挖掘的热点应用

就目前来看，数据挖掘的几个热点包括网站的数据挖掘、生物信息或基因的数据挖掘和文本数据挖掘。下面就这几个方面加以简单介绍。

1．网站的数据挖掘

随着 Web 技术的发展，各类电子商务网站风起云涌，建立一个电子商务网站并不困难，困难的是如何让电子商务网站有效益。要想有效益就必须吸引客户，增加能带来效益的客户忠诚度。电子商务业务的竞争比传统的业务竞争更加激烈，原因有很多方面，其中一个因素是客户从一个电子商务网站转换到竞争对手那边，只需点击几下鼠标即可。网站的内容和层次、用词、标题、奖励方案、服务等任何一个地方都有可能成为吸引客户、同时也可能成为失去客户的因素。而同时电子商务网站每天都可能有上百万次的在线交易，生成大量的记录文件和登记表，如何对这些数据进行分析和挖掘，充分了解客

户的喜好、购买模式，甚至是客户一时的冲动，设计出满足于不同客户群体需要的个性化网站，进而增加其竞争力，几乎变得势在必行。

在对网站进行数据挖掘时，所需要的数据主要来自于两个方面：一方面是客户的背景信息，此部分信息主要来自于客户的登记表；而另外一部分数据主要来自浏览者的点击流，此部分数据主要用于考察客户的行为表现。但有的时候，客户对自己的背景信息十分珍重，不肯把这部分信息填写在登记表上，这就会给数据分析和挖掘带来不便。在这种情况之下，就不得不从浏览者的表现数据中来推测客户的背景信息，进而再加以利用。

就分析和建立模型的技术和算法而言，网站的数据挖掘和原来的数据挖掘差别并不是特别大，很多方法和分析思想都可以运用。所不同的是网站的数据格式有很大一部分来自于点击流，和传统的数据库格式有区别。因而对电子商务网站进行数据挖掘所做的主要工作是数据准备。

2．生物信息或基因的数据挖掘

生物信息或基因数据挖掘则完全属于另外一个领域，在商业上很难讲有多大的价值，但对于人类却受益匪浅。例如，基因的组合千变万化，得某种病的人的基因和正常人的基因到底差别多大？能否找出其中不同的地方，进而对其不同之处加以改变，使之成为正常基因？这都需要数据挖掘技术的支持。

对于生物信息或基因的数据挖掘和通常的数据挖掘相比，无论在数据的复杂程度、数据量还有分析和建立模型的算法而言，都要复杂得多。从分析算法上讲，更需要一些新的和好的算法。现在很多厂商正在致力于这方面的研究。但就技术和软件而言，还远没有达到成熟的地步。

3．文本的数据挖掘

人们很关心的另外一个话题是文本数据挖掘。举个例子，在希赛教育的客户服务中心，把培训顾问与客户的谈话转化为文本数据，再对这些数据进行挖掘，进而了解客户对服务的满意程度和客户的需求，以及客户之间的相互关系等信息。从这个例子中可以看出，无论是在数据结构还是在分析处理方法方面，文本数据挖掘和前面谈到的数据挖掘相差很大。文本数据挖掘并不是一件容易的事情，尤其是在分析方法方面，还有很多需要研究的专题。目前市场上有一些类似的软件，但大部分方法只是把文本移来移去，或简单地计算一下某些词汇的出现频率，并没有真正的分析功能。

随着计算机计算能力的发展和业务复杂性的提高，数据的类型会越来越多、越来越复杂，数据挖掘将发挥出越来越大的作用。

4．数据挖掘在商业上的应用

数据挖掘所能解决的典型商业问题包括数据库营销、客户群体划分、背景分析、交叉销售等市场分析行为，以及客户流失性分析、客户信用记分、欺诈发现等。

基于数据挖掘的营销对我国当前的市场竞争中也很具有启发意义，经常可以看到繁

华商业街上一些厂商对来往行人不分对象地散发大量商品宣传广告，其结果是不需要的人随手丢弃资料，而需要的人并不一定能够得到。如果搞家电维修服务的公司向在商店中刚刚购买家电的消费者邮寄维修服务广告，卖特效药品的厂商向医院特定门诊就医的病人邮寄广告，肯定会比漫无目的的营销效果要好得多。

本章参考文献

[1] 韩中华，吴成东，刘海涛. 数据挖掘技术研究进展. http://www.sz-sws.com/site02/article_show.asp?ArticleID=103

[2] 佚名. 数据挖掘技术概述. http://www.mtg.bjstats.gov.cn/personal/zhdh/d3.htm

[3] 刘红岩，何军. http://diy.ccidnet.com/pub/article/c27_a48704_p1.html

[4] 杨辉. 数据挖掘及其在商业银行中的作用. 中国金融电脑，1998

[5] Jiawei Han, Micheline Kamber. 数据挖掘——概念与技术（影印版）. 北京：高等教育出版社，2001

[6] 数据挖掘讨论组. http://www.dmgroup.org.cn

[7] 彭木根. 数据仓库技术与实现. 北京：电子工业出版社，2002

[8] 徐振航，刘莉芹. 解析 XML 与面向 Web 的数据挖掘技术. http://diy.ccidnet.com/pub/article/c1114_a37466_p1.html

[9] 郑纬民，黄刚. 数据挖掘工具及其选择数据挖掘的流程. http://seekjob.myrice.com/dm-2.htm

[10] 武根友. 数据挖掘与知识发现综述. http://www.dwway.com

第 8 章　操作数据存储

ODS 是围绕一组数据而设计的一种数据结构，即为了特定目的而整合数据。ODS 的特点包含实时模式下动态更新的当前数据。简而言之，ODS 是一个面向主题的、集成的、可变的、当前的（或接近当前的）细节数据集合，用于支持企业对于即时性的、操作性的、集成的全体信息的需求。常常被作为数据仓库的过渡，也是数据仓库项目的可选项之一。

因此 ODS 是用于支持企业日常的全局应用的数据集合，ODS 是企业数据架构中最为复杂的一种形态，既要满足数据事务的操作要求，又要满足数据分析要求，从技术构建角度考察，难度比较大。

8.1　ODS 概述

ODS 是能支持企业日常的全局应用的数据集合，是不同于数据库（DataBase，DB）的一种新的数据环境，是数据仓库扩展后得到的一个混合形式，是一个集成了来自不同操作数据库数据的环境。其目的是为终端用户提供一致的企业数据集成视图。它可以使用户轻松应对跨多个商业功能的操作挑战。作为一个中间层次，它算不上事务处理，也算不上高层决策分析。和数据仓库和数据集市受约束的更新途径相比，它的关键差异在于应用程序的更新频率和直接的更新途径。

8.1.1　ODS 的特点

ODS 具有如下特点。

（1）面向主题的：ODS 不仅被设计用来满足公司重要主题的需求，而且也可以满足特定功能或应用程序的要求（如风险管理、获得客户的全部数据视图等）。

（2）集成的：现有系统通过一个转化过程把详细数据导入到 ODS 中去，这样可以获得一个集成的、公司范围知晓的数据库。这个转化过程和 DW 的转化过程很相似。当处理多个现有系统时，经常会出现数据识别和一致性的问题，这些问题必须得到解决，如在识别客户时要描述人的性别，不同的系统可能采用不同的代码。

（3）可变的：存取的数据可以联机改变，包括增加、删除及更新等操作。

（4）当前的（或接近当前的）：ODS 中的数据是不断地被更新的。数据一直保持变化需要很高的更新频率，这些变化要求基础的现有系统尽快满足 ODS 对速度方面的要求，也就是说，速度很重要。有时往往要求立即得到一些数据，另一些数据在每天更新

时得到。这样，ODS 的更新必须是快速和高频的。频率是指在数据很可能来自完全不同的现有系统条件下，通过特定的迁移过程，ODS 被更新的次数。当然，要将发生更新的数据量考虑在内；速度是指一次必须发生的更新从一个现有系统发生变化的那个时刻到它反映在 ODS 中那个时刻的速度。

（5）当前数据：一个 ODS 反映了它下面源数据系统的状态。数据库完全是最新的，有足够的数据来说明当前及以后的情况，除非出现极其重要的商业请求才会改变这条规律。如果 ODS 必须包含历史数据，就必须确信对哪些是所需的历史信息和所需原因完全了解，而且还必须考虑到衍生问题，如规模估计、归档和执行。

（6）详细：ODS 被设计为统一操作服务，所以要坚持详细的标准。详细的定义依赖于 ODS 正在解决的商业问题。数据粒度可能与源操作系统相同也可能与源操作系统不相同。例如，在一个单独的源系统中，每个单一账户的平衡十分重要；但对于使用 ODS 的职员来说，重要的可能是所有账户的综合平衡。

8.1.2　ODS 的作用

一般在带有 ODS 的系统体系结构中，ODS 都设计为如下几个作用：

（1）在业务系统和数据仓库之间形成一个隔离层。一般的数据仓库应用系统都具有非常复杂的数据来源，这些数据存放在不同的地理位置、不同的数据库、不同的应用之中，从这些业务系统对数据进行抽取并不是一件容易的事。因此，ODS 用于存放从业务系统直接抽取出来的数据，这些数据从数据结构、数据之间的逻辑关系上都与业务系统基本保持一致，因此在抽取过程中极大降低了数据转化的复杂性，而主要关注数据抽取的接口、数据量大小、抽取方式等方面的问题。这种运用有时被称为数据准备区（Data Staging Area，DSA）。

（2）转移一部分业务系统细节查询的功能。在数据仓库建立之前，大量的报表、分析是由业务系统的数据库直接支持的，在一些比较复杂的报表生成过程中，对业务系统的运行产生相当大的压力。ODS 的数据从粒度、组织方式等各个方面都保持了与业务系统的一致，那么原来由业务系统产生的报表、细节数据的查询自然能够从 ODS 中进行，从而降低业务系统的查询压力。

（3）完成数据仓库中不能完成的一些功能。一般来说，带有 ODS 的数据仓库体系结构中，DW 层所存储的数据都是进行汇总过的数据，并不存储每笔交易产生的细节数据，但是在某些特殊的应用中，可能需要对交易细节数据进行查询，这时就需要把细节数据查询的功能转移到 ODS 来完成，而且 ODS 的数据模型按照面向主题的方式进行存储，可以方便地支持多维分析等查询功能。

在一个没有 ODS 层的数据仓库应用系统体系结构中，数据仓库中存储的数据粒度是根据需要而确定的，但一般来说，最为细节的业务数据也是需要保留的，实际上数据的内容也就相当于 ODS，但与 ODS 所不同的是，这时的细节数据不是“当前、不断变

化的”数据，而是“历史的，不再变化的”数据。ODS 可以和 DW 形成互补的整体，构成完整的战术决策支持系统架构。利用 ODS＋DW 实现战术决策支持有其非常直观的优势：利用 ODS 实现实时或准实时的数据抽取，而且 ODS 的数据量不大，可以比较高效地进行数据的修改和更新，并且可以提高查询的效率。而利用数据仓库的海量存储实现历史数据的查询，实现战略决策支持。

但是，这种方式也有很明显的劣势：由于 ODS 和 DW 的结构和模型是不同的，这需要进行不同的系统和数据模型设计，也需要不同的系统维护过程，相应增加系统的使用成本。

8.1.3 ODS 的分类

按照数据从数据源载入 ODS 系统的频度不同，可以将其划分为三类，分别是 Class-I 型 ODS、Class-II 型 ODS 和 Class-III 型 ODS。

Class-I 型体系结构以同步方式将信息加载到 ODS 中，这种载入方式使得来自于多个源数据的数据变换很有限，主要完成的数据抽取工作是大量的数据集成，因此对它的源系统要求有较为统一的数据格式。这一结构的 ODS 主要适用于需要集成业务处理平台的环境中，如信贷业务中的客户贷款服务，它涉及的一般不仅仅是简单的前台操作，还涉及到银行资金库存周转情况、客户信息、客户提款等活动。这一结构有利于提高效率及准确度。

Class-II 型体系结构以存储转发方式将数据加载到 ODS 中，它的更新频率一般为数小时一次或更频繁。其数据包括较多的集成信息，因此，可提供功能强大的用户视图和统一报表、汇总表分析，有助于进行日常的决策分析。该结构适用于需要进行战术性决策支持的企业。

Class-III 型体系结构的数据加载方式如同数据仓库中细节性数据的抽取工作，数据更新周期通常为日加载。它可以作为基于 DB-ODS-DW 三层结构的决策支持系统的一部分，因此它可以辅助日常的决策支持，也可以将对数据仓库基于细节数据的查询处理放在该层中实现。它有助于实现一个较全面的决策支持系统，既能满足面向中层管理人员的日常分析和决策，也可以满足企业长远的战略性决策。

8.1.4 ODS 和 DW 的联系与区别

面向主题和集成性使得 ODS 的数据在静态特征上很接近 DW 中的数据，但事实上 ODS 和 DW 完全不同，具体表现在以下几个方面。

1．当前数据

ODS 是数据的当前视图，包含了当前和近似当前的数据。它可能会精确到以秒计算，但这并非必须满足的标准，具体要看 ODS 的目标、用户需要和对速度的商业需求。例如，一个投资公司的风险管理 ODS 可能只会更新一次海外市场的情况。如果商业环境

发生变化，例如从库存中取出一部分零件，那么和该环境相关的操作数据也会随之改变。ODS 反映了商业环境的当前状况，在本例中即是库存零件的数量。

DW 中的数据是长期保存并可重复查询的数据，它更多地反映了商业环境的历史视图。数据如果发生了变化，会生成一个快照并插入这个商业环境的其他快照队列中。DW（快照、事件）的插入一般是批量进行的，仅根据预定义的时间表进入 DW。所以，DW 并不能反映即时的操作状态。

ODS 通过提供当前或准实时的集成信息，满足组织机构日常的操作数据要求；而 DW 通过提供更多的历史信息，支持组织机构的情报和战略要求。

2．更新/载入数据

在 ODS 中，因为只用一个状态反映一个商业环境，所以经常会更新（插入、修改、删除）现有的数据记录。在 DW 中，并不更新记录，而是可能创建一个新的快照或事件。

更新数据时的另一个不同点在于结束更新的时间点。根据商业需求，ODS 更新得越快越好，这意味着可能需要一个实时或准实时的更新机制。这是 ODS 显著的特征之一。而 DW 的更新通常是集中进行的，通过预先确定的时间点向 DW 中载入数据。

3．数据汇总

ODS 包含了主要的详细数据，有时出于性能原因也包含综合和汇总数据，但这些综合和汇总只有在需要时才生成，并且只有在生成时是准确且不可重建和保存的。

因此，汇总数据通常根据要求进行计算。由于它的有效周期很短，也将其称为“动态汇总数据”，其数值依赖于计算的时刻。例如，假设一个公司的整个账户余额由公司所有单个账户计算汇总而得，如果任一单个账户数值发生变化，则汇总也会相应变化。

但是，ODS 中可能会保留某些经常使用的汇总合计。与之不同的是，DW 中既保存细节数据，也保存汇总数据。它的汇总数据被称为“静态汇总数据”。这些汇总数据的结果并不依赖于计算的时刻。例如，假设希赛教育的一个经理想要计算 2008 年度视频课程的销售总量。如果在 2009 年 2 月计算该数值，将和在 2009 年 1 月得到的结果相同（假设在 2009 年 1 月之前已经载入了 2008 年 12 月的数据）。

4．数据建模

ODS 是为记录级别的访问而设计的，DW 或数据集市则是为结果集的访问而设计的。ODS 支持快速数据更新，DW 则完全是查询驱动的，根据商业需求快速而直观地提供给人们数据。但是，体系结构中 DW 的单个片段可能不是这一类型的。

例如，商业数据仓库是一个标准数据存储的企业数据仓库。它优化了数据输入和数据抽取，这样的数据仓库应该实施更标准化的结构。它们也用于需要访问最基层信息和全部历史的大量复杂查询。而商业信息仓库（数据集市结构）优化了对数据出于商业目的的访问。所以可以明显地看到通过应用星型模型作为一种物理机制来达到性能要求。维数模型是商业用户查询数据的一种简单方法。虽然并没有为长时间运行的查询做优化，但用户可以简单而快速地访问大量数据。它依靠物理实施来体现性能。

因为 ODS 通常由应用软件独立设计，所以规定了 3NF（第三范式）数据模式。

5．查询和交易

ODS 中的交易量很小，几乎不耗费资源，有确定的完成速度。大量交易通常是并行进行的，一天中会频繁完成一些相同的交易。在 DW 中，查询或分析工作量较大，消耗的资源很多，完成的速度也不确定。

举例来说，一个操作过程可以得到一个客户的所有账户和余额的列表。而 DW 工作单元可以计算出银行全部存款总额和所有客户抵押总额之间的差别。

6．使用

ODS 用于职员或企业的日常决策。例如，在保险公司，代理人通过访问客户的合同来决定还能提供哪些其他的产品。和 ODS 不同的是，DW 服务于决策支持系统分析员或高级决策层。保险公司使用 DW 用于战略的、长期的和公司范围的决策。例如，公司可能决定为去年销售量较低的某些产品举行市场推广活动。

ODS 是日常性的，可用于即时（up-to-the-second）决策，而 DW 是战略性的，用于长期趋势分析或战略决策。

7．用户

由于 ODS 和 DW 完成的任务不同，因而使用这些系统的用户群体也不同。

ODS 适用于日常用户。这包括诸如保险公司代理人这样的用户，需要每天联系客户并随时得到信息。DW 则更多地用于战略用户群，他们需要制定战略决策。例如，一个保险公司负责寿险部门的经理想得到近 5 年来保险增长总量的数据。

8．数据量

数据量是主要的区别之一。ODS 包含的数据量比 DW 少得多。

ODS 可能包含了每个客户余额（按产品分类）的当前值，以反映一个客户在公司范围的总余额。DW 则可能存储了每个客户余额的历史快照。

ODS 将包含单个订单系列条款的明细，而 DW 则可能仅有订单的标题。

ODS 可能为市场、会计和采购部门存储了客户的联系方法，DW 则可能存储了过去 5 年内客户的联系方法。

9．总结

表 8-1 总结了 ODS 和 DW 之间的各种不同之处。

表 8-1　ODS 和 DW 之间的不同

ODS	DW
数据很详细，具有可靠的可用性	数据可能不很精确，但是足够为战略分析服务；数据不必有很高的可用性
包含当前的和准实时的数据	包含历史数据
实时和准实时的数据载入	通常是批量数据载入
大多数更新是字段级别的（甚至可能是附加的）	数据是附加的，并不更新

续表

ODS	DW
仅有明细的数据	包含汇总和明显的数据
建模支持快速数据更新（3NF）	运用各种数据建模技术，特别是用于 DW 的 3NF 和用于数据集市的维度模型来优化查询性能
和一个 OLTP 系统很相似	查询过程往往需要处理更大的数据量
用于日常决策的制定和操作报告	用于长期决策的制定和管理报告
用于操作级别	用于管理级别

DB、ODS 和 DW 三者之间的性能如表 8-2 所示。

表 8-2　DB、ODS 和 DW 三者之间的性能比较

特　　性	DB	ODS	DW
用户对象	业务员	业务分析员	经理、分析员
数据内容	当前、实时	当前或接近当前	历史
数据访问	单个记录、事务驱动	单个记录、事务或分析驱动	多组记录、分析驱动
数据粒度	细节级	细节和轻度综合	综合与分割
数据冗余	系统内非冗余系统间冗余	有操作数据冗余	有管理数据冗余
数据组织	操作型	面向主题	面向主题
数据更新	频繁、即时	大量、非即时	不可更新
数据规模	小到中级	中级	较大
业务支持	支持日常操作型	支持日常决策和操作	支持企业管理分析
响应时间	微秒级	秒到分钟级	秒到小时级

ODS 与传统的 DB 不同之处还在于 ODS 提供了全局一致的 OLTP，而 DB 只适用于部门级的 OLTP。

8.1.5　从 DB 向 ODS 转化的实现机制

在 DB-ODS 的体系结构中，ODS 的实现机制表现在其记录系统定义的数据传送关系上。操作型环境中各分散的 DB 记录经过过滤后形成了 ODS 系统的记录系统，向 ODS 系统中提供数据。记录系统定义了原有分散 DB 中哪些数据送往 ODS，并指明与 ODS 数据相应的数据表。通过 ODS 的定义可以把分散于应用的 DB 中的数据复制到 ODS 中去，这样原来的分散 DB 中的记录就形成了 ODS 中的面向主题的记录。ODS 维护着一个分析型的环境，数据处理简单得多，实际需要的支持技术也较少。

至于从 ODS 向 DB 转化的实现机制，这种情况主要用在有关企业全局操作应用的情况，可以通过在 ODS 系统中存放一些参数表，它所反应的关系是 ODS 全局更新时必须要反应到所有 DB 中的相关记录。此时，ODS 是一个操作型环境，实现 ODS 所要求的技术跟原来的面向应用的分散的数据库系统一样，包括事务管理、封锁管理、数据恢复

等技术。

8.2 ODS 的应用

ODS 应用主要有两方面：一类是作为一种独立的解决方案为需要集成操作环境的企业提供全局一致的应用；另一类是作为从 DB 向 DW 的一种过渡形式，建立一种基于 DB-ODS-DW 的三层体系结构的决策支持系统解决方案。

在开发全局应用的 ODS 系统时需要的支持技术包括事务管理、封锁管理、死锁检查、数据恢复、日志管理及数据存储管理等类似于数据库开发时的复杂技术。在 ODS 上开发全局应用有如下优点：

（1）由于 ODS 的信息是全局一致的，所以在进行全局应用时无须再进行数据集成，这将大大提高全局应用的处理效率。

（2）各部门的数据库也可以进行全局应用。另外 ODS 支持即时 OLAP。

ODS 用于决策支持系统中，确实给决策支持系统带来了新的特点和优点，主要体现在如下几个方面：

（1）ODS 为 DW 的建立提供了一致的数据接口环境以供抽取，ODS 的存在承担了 DW 的一部分工作，从而减轻了 DW 的负担，使 DW 的管理工作变得较为简单。

（2）由于 ODS 中的数据量远小于 DW，加之它的开发周期较短，所以对中小企业来说承担的风险也较小，可以在较短的时间内得到应用并取得经济效益。

（3）ODS 的建立弥补了 DB-DW 两层体系结构中所存在的不足，它可以满足数据处理的多层次要求，即面向各级管理人员，更加有效地利用信息资源，为企业决策分析服务。

尽管创建 ODS 系统可以采用数据仓库技术，但由于通常 ODS 既要执行传统的事务处理环境的功能又要执行特定决策支持环境的功能，而这两种环境本身差异很大，充分发挥这两种环境性能的技术是直接冲突的，因此 ODS 的设计比较复杂。在考虑选取哪种类型的 ODS 应用、体系结构采用集中式的还是分布式的、开发方法采用“自上而下”的方法还是“自下而上”的方法，这些因素都与企业自身的性质及状况、企业规模的大小、经营业务范围等具体情况紧密相关。

系统设计人员的很多任务都是在有冲突的需求之间进行处理的，以及预测这些需求如何因时间而可能变化。需求包括响应时间、吞吐量、开发简化、成本、服务水平、资源利用、可用性、容错、可服务性等。可提供 ODS 类型功能的扩展应用数据库很有可能受到约束，因为它的设计初衷是应用软件特定数据库，这可能会限制其性能。如果为支持 ODS 功能而改变 ODS，就有可能反过来影响原有的应用程序。

8.3　ODS 系统的设计

ODS 系统的设计涉及到很多相关的技术问题，可将其划分成以下几个部分：ODS 数据转换层、ODS 平台和 ODS 中间件。

8.3.1　ODS 数据转换层

数据转换层由各种转换工具组成，主要完成从源数据系统到 ODS 系统的数据转换、净化和载入。转换工具编写的好坏是建立一个成功的 ODS 系统的关键所在。转换工具应具有下述功能：

（1）支持大量源和目的平台。这一功能要求数据转换工具能处理各种指定的源系统中的数据，包括对这些数据类型及数据变换的理解。转换工具的设计应与 ODS 平台有很好的兼容性，同时它还应具有很好的可移植性。

（2）对变化进行捕捉的能力。在 ODS 系统中实现同步的关键在于能够及时有效地捕捉到操作环境中的变化。对于时间不敏感的 Class-I 型 ODS 可以通过读取日志磁带记录来实现，而对于 Class-I 型 ODS 就必须在事务变化的同时捕捉变化并将这种变化直接传送给 ODS 系统，实现起来较为复杂。

（3）支持多种类型的转换。这点要求类似于 DW 系统，即要求该层至少能提供支持集成、解码、聚集等转换功能。不同于 DW 的是 ODS 中的转换工具要在适当的位置进行数据的更新操作，另外对于像 Class-I 型这类对时间敏感的 ODS 系统，还要求转换工具具有能从变化捕捉程序中获得数据并将数据直接传送到 ODS 中的功能。

（4）对设计方法的变化具有透明性。由于 ODS 系统开发类似于 DW 的系统开发，是一个反复循环的过程，设计不是一成不变的，因此一个好的转换工具不应随着设计方法的改进而变得一无用处，可通过建立规则引擎机来实现对转换层工具的调整使其随着设计的改变自动进行调整，而不需要对转换工具加以变更。

（5）提供数据的可靠传输。转换层务必确保来自于操作环境的数据源能可靠传送到 ODS 系统中，这是保证 ODS 中数据与操作环境中数据一致的一个重要方面。

（6）对元数据的支持。这一功能要求转换层对 ODS 系统中的元数据能进行维护。元数据中包含了 ODS 系统当前状态信息和关于创建 ODS 系统规则的信息，这些信息对被抽取的数据源进行定位并为这些数据加上时间信息。由于用户无法直接利用存在于元数据中的信息，因此转换工具应能提供一种开放的和用户可以访问的途径来访问元数据并对其进行管理。

8.3.2　ODS 平台特性

对 ODS 运行的软、硬件平台有很多要求，这些要求根据所选取的 ODS 类型而不同。

Class-I 型 ODS 主要提供集成操作环境，其实现技术类似于数据库系统。下面以 DSS 环境中的 ODS 系统为例，其平台特性提出的要求包括如下几个方面。

（1）规模伸缩性。这一功能主要是对用于决策支持环境中 ODS 系统要求的，因为在这一应用中，采用的设计方法类似于 DW 的设计方法。这种方法不同于传统的数据库设计方法，系统的需求分析并没有在开始创建 ODS 或 DW 时结束，而是贯穿于系统的整个开发设计甚至应用过程当中，因此要求 ODS 平台有很好的规模扩展性，来满足系统设计变化的需求。

（2）对多种大型网络环境的支持。基于 DSS 的 ODS 应用是建立在传统的 DB 应用系统之上，提供网络支持，一方面各层用户能对原有系统以及对高速开放的局域网和广域网的访问，另一方面也有利于应用的扩展和信息的收集。

（3）事务监视管理功能。尽管在 DBMS 层中提供了事务监视功能，但却不能从高层对事务进行有效控制和管理，同时 ODS 层既有操作环境又有分析环境并且还是各层用户访问的公共接口，因此有必要在 ODS 层提供事务监视管理功能，保证系统资源的合理利用及可靠运行。

（4）处理大量查询的能力。ODS 平台是一种集事务处理、复杂查询优化处理及大量数据管理等多种功能于一体的平台系统，尤其是要求它能实现高性能查询处理。对那些不能通过索引方式进行访问的数据，采用传统的分类、合并联接技术扫描查找，将会浪费数据库资源，同时也很费时，采用多索引技术，对多索引的创建和维护又需要占用大量的 CPU 资源，造成资源紧张，因此在 ODS 系统中使用并行处理器将会大大提高整个系统的性能。ODS 平台要求可执行大量的连接、合并和聚集操作，还能支持传统的散列法和分类法在其上的应用。

（5）提供混合作业处理的能力。在 ODS 系统中的作业包括：面向操作及办公处理的短时间的简单查询任务，面向决策分析人员的长时间的复杂查询任务及与操作环境同步处理插入、更新和删除操作时的一致性处理任务。ODS 平台应提供对这些作业混合运行的处理，还有对查询优先权进行处理的能力。例如，长时间低优先权的查询作业应能被短时间高优先权的作业、更新作业中断处理；有些平台还提供这样一种功能，将那些对处理时间要求不高的查询作业安排在夜间执行、建议用户推迟执行一些大型查询作业并降低它们的优先权。

（6）分布数据的一致映射。是指对分布在多个物理场地的数据源，在使用时使用户感觉不到它的分布性，逻辑上数据好像来自于同一个场地，这种对分布数据的透明访问在一些 DBMS 技术中已经得到了应用。

8.3.3 ODS 系统中间件

ODS 系统中间件的一个关键组成部分就是网络中间件。中间件的主要作用是提供用户和 ODS 系统数据之间的无缝连接，同时还提供一系列应用程序接口允许应用程序同

本地或异地 ODS 系统进行通信。另一方面，在 DB-ODS-DW 三层结构中，中间件也是将 ODS 中的信息与 DW 系统中信息相关联的一条途径。具体要实现的技术包括以下几个方面：

（1）对多种类型客户接口的支持。要求 ODS 中间件能提供多种类型的工具和用户应用程序。由于用户使用的可能不是流行的 PC 和苹果 Mac 机，因此，要提供多种用户机的接口，以支持多种用户的连接访问。

（2）提供对多种关系型及非关系型数据库的兼容性。中间件是用户与 ODS 系统数据无缝连接的一个接口，而 ODS 系统数据很可能来源于多种关系型数据库或非关系型数据库，因此要求中间件能提供对多种数据库实现透明访问的机制。

（3）提供监视及收集统计数据的功能。因为 ODS 中间件是所有查询活动要访问的一个公共接口软件，由它来协助管理 ODS 系统可以带来很大的方便性，通过它的监视功能可以有效地平衡系统资源的利用和及时协调故障恢复。在这里中间件若能将 ODS 中存放的大量静态数据进行关联收集，必将很好地为决策支持服务。另一方面，由中间件软件来实现 ODS 系统安全性及一致性维护，也是一种很好的方案。

（4）提供查询作业排队及计划管理功能。对整个系统的查询任务进行合理化的全局管理是中间件提供的一个最为重要的功能。在对查询任务进行排队时，当队列长度超过某一阈值，就由中间件向用户发出消息，建议用户将查询任务安排在非高峰期间执行，或对查询要求自动进行调整处理，也可将一些查询结果延迟返回，以缓解资源紧张程度。

8.3.4　ODS 系统数据建模

当企业数据模型存在时，可从该模型导出 ODS 数据模型。企业数据模型描述了公司在收集信息方面所需要的所有数据，ODS 数据模型是企业数据模型的子集，仅为 ODS 数据模型提取真正需要的那些部分。

有两种不同的数据建模方法可以满足 ODS 的需要，分别是实体关系（Entity-Relation，E-R）模型和维度建模。

1．E-R 建模

由于 E-R 模型可用于理解和简化商业领域和复杂系统环境中的模糊数据关系，因此它是一种抽取工具。E-R 建模方法可使用实体和实体之间的关系，产生特定兴趣领域的数据模型。

（1）实体。实体可定义为人、地点、事情，以及商业或组织的相关事件。例如，产品、订单等。实体代表一类对象，它们是现实世界中可以按属性和特征进行观察和分类的一些事物。

（2）关系。关系描述模型中各实体之间的结构性交互和关联。它显示了实体间的相关性。例如，产品与订单之间的关系。

大多数 ODS 实施 3NF 模型。这类模型最初是为最小化数据冗余而设计的，因为该

模型在值发生改变时，可使数据库中的更新数量达到最小。

2．维度建模

维度建模是一种将数据模型概念化和形象化为一组可用一般商业概念描述的度量的技术。在总结和重新整理数据以及显示数据视图以支持数据分析时，该技术特别有用。维度建模主要处理数字数据，比如值、计数、重量、平衡和出现次数。维度模型的基本概念为事实、维度、度量（变量）。

（1）事实。事实是相关数据项的集合，包含度量和环境数据。每个事实一般代表商业项、商业事物或可以在商业或商业过程分析中使用的事件。

（2）维度。维度是从特定角度描述事实数据的一组成员或单位。在图表中，维度通常是用轴来表示的。在维度模型中，事实表中的每个数据点都与多维中每个维度的一个成员相关联。维度决定了事实的环境背景。

（3）度量。度量是事实的数字属性，表示商业相对于维度的性能和行为。其实际成员称为变量，如销售额、销售量、供应量等等。度量由维度的成员组合来决定，并定位到事实中。

维度建模的基本模型是星型模型，模型通常有一个较大的中央表（事实表）和一组以放射状围绕在事实表周围的较小的表（维度表）。

3．确定建模技术

两种数据建模技术有时看上去有很大的不同，但它们也有很多相似之处。维度建模可以使用相同的符号，比如实体、关系、属性和主关键字等。而且，通常可以说事实就是一个实体，其主关键字为外关键字的组合，而外关键字又引用维度。因此，可以说维度建模是 E-R 建模的一种特殊形式。但是，传统的 E-R 模型有着平均而平衡的实体风格以及实体间复杂的关系，而维度模型却非常不对称。

优先使用哪种技术取决于 ODS 的目的。但一般来说，如果 ODS 用作单纯的业务处理，则优先使用 E-R 建模技术。在业务处理中，通常只是对非常特殊的任务请求少量数据。这些任务一般由 E-R 模型的实体来表示。当然，在业务处理中，同样需要来自源系统的非常快的数据。

由于 E-R 模型中没有数据或只有少量的冗余数据，因此可以从源系统或通过应用程序非常快地进行更新。快速更新功能的另一个原因是现有系统通常也是使用 E-R 模型设计的。因此，在更新过程中的转换需要可能会非常小。

如果 ODS 更多地用作数据访问系统，那么就应该优先使用维度建模方法。在这种情况下，通常已安排了数据更新。在更新数据的过程中，可进行许多转换，并可安排数据以满足数据访问应用程序的需要。这也意味着更高的数据冗余，可使用于分析的复杂查询更快速地进行。

成功数据模型的基础是对业务需求非常准确的分析。根据业务问题的情况，建模的中间形式可能会多种多样。如果 ODS 要实现多个目的，则必须决定针对每个目的使用

哪种技术。就 ODS 的物理数据库设计而言，不需要特殊的技术。可以使用所有已知技术（例如，以物理方法合并表、使用索引或定义清除条件等）从 ODS 中抽取数据。

8.3.5　ODS 系统设计步骤

本节以维度模型为例，介绍 ODS 系统的设计步骤。

1．确定数据范围

确定数据范围实际上是对 ODS 进行主题划分的过程，这种划分是基于对业务系统的调研的基础上而进行的，并不十分关心整个系统上端应用需求，但是需要把上端应用需求与 ODS 数据范围进行验证，以确保应用所需的数据都已经从业务系统中抽取出来，并且得到了很好的组织。一般来讲，主题的划分是以业务系统的信息模型为依据的，设计者需要综合各种业务系统的信息模型，并进行宏观的归并，得到企业范围内的高层数据视图，并加以抽象，划定几个逻辑的数据主题范围。在这个阶段，以 E-R 模型表示数据主题关系最为恰当。

2．数据分析和主题定义

第一步定义出了企业范围内的高层数据视图以及所收集到的各种业务系统的资料，在这一步中，需要对大的数据主题进行分解，并进行主题定义，直到每个主题能够直接对应一个主题数据模型为止。这个阶段将把第一步生成的每个 E-R 图中的实体进行分解，分解的结果仍以 E-R 模型表示为佳。

3．定义主题元素

定义主题元素包括定义维、度量、主题、粒度、存储期限。

（1）定义维的概念特性。包括维名称、维成员和维层次。其中维名称应该能够清晰表示出这个维的业务含义，维成员是这个维所代表的具体的数据，维层次是维成员之间的隶属与包含的层次关系，每个层次需要定义名称。

（2）定义度量的概念特性。度量名称：名称应该能够清晰标出这个度量的业务含义。

（3）定义主题的概念特性。主题名称和含义：说明该主题主要包含哪些数据，用于什么分析；主题所包含的维和度量；主题的事实表以及事实表的数据。

（4）定义粒度。主题中事实表的数据粒度说明，这种粒度可以通过对维的层次限制加以说明，也可以通过对事实表数据的业务细节程度进行说明。

（5）定义存储期限。主题中事实表中的数据存储周期。

4．迭代，归并维、度量的定义

ODS 中数据来自于多个系统，数据主题划分时虽然对数据概念进行了一定程度上的归并，但具体的业务代码所形成的各个维以及维成员等还需要进一步进行归并，把概念统一的维定义成一个维，不允许同一个维存在不同的实体表示。

5．物理实现

定义每个主题的数据抽取周期、抽取时间、抽取方式、数据接口、抽取流程和规则。

物理设计不仅仅是 ODS 部分的数据库物理实现，除设计数据库参数、操作系统参数、数据存储设计之外，有关数据抽取接口等问题也必须清晰定义。

本章参考文献

[1] Corinne Baragoin. 在 DB2 通用数据库上建立操作数据存储. IBM 公司红皮书，2001

[2] 林宇. 数据仓库原理与实践. 北京：人民邮电出版社，2003

[3] 李明. ODS 系统的应用与设计. 计算机工程与应用，2002

[4] W H Inmon. Building the Operational Data Store. New York: John Wiley & Sons,Inc, 1996

[5] 王珊. 数据仓库技术与联机分析处理. 北京：科学出版社，1998

[6] Joyee Bischoff, Ted Alexander. 数据仓库技术. 北京：电子工业出版社，1998

第 9 章　企业应用集成

20 世纪 90 年代，企业资源计划（Enterprise Resource Planning，ERP）应用开始流行，由于它们需要支持已有的应用和数据，因而对企业提出了新的问题。同时，信息技术和网络技术的飞速发展，也为企业对内和对外优化业务流程管理，实现数据流和业务运作的自动化，以及在 Internet 上开展电子商务提供了强有力的工具。企业应用集成（Enterprise Application Integration，EAI）正在是这种环境下应运而生的，并且一直处于高速发展之中。

EAI 涉及管理技术、信息技术等诸多领域的一门新兴技术，而且还处于发展的初期，因此还存在着许多问题。其实，Enterprise 理解为“企业”并不完全准确。它所代表的资源集成的管理思想和所用的技术、方法可用广泛应用于政府部门、事业单位等非盈利组织的信息化环境。因此，在下面的讨论中，并不限于日常所说的“企业”，但为尊重习惯，还是把 EAI 称为企业应用集成。

9.1　EAI 概述

所谓企业应用集成，是指在企业范围内，将多个应用系统的过程、软件、标准和硬件集成起来，使其成为无缝运作的整体。从范围来看，EAI 既包括在企业内部进行的系统集成，也包括企业为实现与外部环境的交互而进行的集成，如企业的 B2B 建设等。

与 ERP 的出现和发展一样，EAI 也是企业管理思想发展的结果。从物料需求计划（Material Requirements Planning，MRP）、MRPII、ERP 乃至所谓的 ERPII，反映了生产企业中管理的范围的逐步扩大——从核心的生产部门，到包括物料、财务、配送、采购等部门，再到包括供应链上完成协同商务的其他企业和客户，等等；在管理范围扩大的同时，信息技术所运用的层次也在不断提高：早期的信息技术体现在部门级的事务处理系统中，伴随着物流完成企业的基本操作；然后是面向特定管理职能（例如，财务、人力资源等），完成特定管理信息的收集、加工和传输；最后是企业范围的各种信息系统的无缝集成，以满足社会经营环境对信息的需求。

与 ERP 等信息系统不同的是，EAI 并不是一种有特定功能的成型的信息系统，而只是运用了特定的、标准化技术对遗留（Legacy）资源的利用和遗留系统的无缝整合。EAI 代表了一种持续集成的信息化战略。对于一个组织来说，EAI 可能是一项长期的、不断进行的工程。

9.1.1 谁需要 EAI?

首先，企业需要 EAI。在今天的网络信息和经济一体化时代，全球企业正向着电子商务的运营模式转变。企业内部各经营环节以及企业与外部连接都非常紧密，甚至融为一体。反映在企业与客户、合作伙伴、供应商、分销商等进行的商务活动更加密切，而且影响着企业内部的作业流程。业务的信息在比以前广泛得多的范围内流动。在这种条件下，企业的信息设施不再是一个个不相干的封闭系统，而必须在内部进行信息整合和流程优化，同时遵从外部的信息标准要求，使企业做成为一个无缝的、高效的、流程优化的社会化的信息服务单元。这就要求整个企业的信息设施（包括硬件、软件、数据、知识等）构成一个无缝、联动的整体。事实上，由于技术发展的渐进性，在企业经营的历史当中，遗留了很多的信息资源，包括落后的硬件、封闭的软件和冗余、不合标准的信息，以及被割裂的处理过程等。因此，必须以一定的标准和流程来对这些遗留资源进行集成或重组，即 EAI。

其次，政府需要 EAI。政府在信息化方面的职能是组织、规范和推动社会的信息化资源，并为市民、企业等提供基于网络的职能服务。政府在社会的信息化过程中的主要作用是规范、引导和集成社会信息资源，同时把自己的服务职能以简单、统一、标准的方式呈现给社会。从这个意义上来说，电子政务或电子政府（e-Government）就是一项迄今为止最大的 EAI 工程。

为进一步界定 EAI 的概念范围，下面把 EAI 与 ERP、Portal（门户）作一比较。

EAI 与 ERP 等系统并没有必然联系。首先，ERP 是适用于企业，尤其是生产型企业的现代管理信息系统，而 EAI 的思想、方法和技术更具有普适性。不管是否实施 ERP 等项目，EAI 都可以作为一种长期的信息化战略。即使对于企业来说，不建设 ERP，也可以进行 EAI，以获得竞争优势。而在 ERP 的建设中，有的 ERP 是要求和支持 EAI 的，有的则不支持 EAI。

EAI 与 Portal 则有天然的联系。Portal 是指基于 Web Service 技术、经过后台业务整合的单一访问界面，是 EAI 的一种主要的技术表现形式。因此，EAI 并不是看不见摸不着的东西，EAI 的实施成果，除了一些遗留系统的升级、新功能的部署以外，主要就是提供一个统一的服务访问门户。

9.1.2 EAI 的内容

EAI 包括硬件、体系结构、软件和过程的集成。它包括以下三个层次。

（1）业务过程集成（Business Process Integration，BPI）：为实现整体的业务目标，要定义、关联和管理不同的业务过程，并通过相应的业务信息系统中实现所需要的信息交换，从而降低成本，更高效地实现客户目标。BPI 的要素包括过程管理、过程建模和工作流。

（2）应用集成（Application Integration）：这一层次的集成目的是将多个应用系统进行绑定，使之像一个实时运行的系统一样接收信息输入和产生数据输出，实现多个系统功能的叠加。应用集成广泛用于 B2B 集成、在后端服务应用基础上建立的客户关系管理（Customer Relationship Management，CRM）系统、集成多个应用的 Web 门户等。在 ERP 应用实施后，也要经常进行与新的应用系统集成。

（3）数据集成（Data Integration）：数据集成是应用集成和业务过程集成的基础。在集成以前，要对数据进行统一标识、分类，并进行元数据建模。这三个步骤完成后，就可以实现企业范围的数据共享和数据分布了。

除此之外，EAI 还有两个重要的技术要素。

（1）集成的技术标准：为实现完全的应用集成，必须选择一个统一的集成标准。常见的集成标准包括 COM+/DCOM、CORBA、电子数据交换（Electronic Data Interchange，EDI）、Java RMI 和 XML。

（2）应用集成的平台：为完成系统集成，必须在异构的网络上实现基础的硬件、软件和体系结构的集成。集成平台就是实现 EAI 的基本设施。它通常由一些专业的软件厂商提供，包括应用服务器，实现某一标准的集成框架，辅助数据、工作流建模和应用开发的工具集，保证数据互通、事务可靠、信息安全、工作流整合的中间件，以及相应的管理工具。应用集成平台既有松散的、免费的工具集，也有专业厂商推出的大型套件。

9.1.3　EAI 的技术基础

EAI 问题是企业经过一段时间的信息化后普遍需要解决的问题。EAI 不仅是企业面临的一个技术问题，还是一个企业面临的管理问题和组织问题，如果单纯考虑企业应用系统集成，很容易使企业步入“IT 黑洞”。随着企业对应用系统集成的实施、开展和逐步的成功，企业的市场反应速度、客户服务质量、组织结构质量都会响应得到提高和加强。EAI 的成功实施必将提高企业的过程敏捷性，为企业信息门户的组建和向 Web Service 发展提供一个坚实的基础。

可以肯定地说，如果没有 Web 为基础的网络应用技术，就没有今天的大规模应用集成。其实，点到点的应用集成早在 Internet 出现前就有了。只有在以 HTTP 为基础的 Web 出现后，不同的计算资源才真正有了国际通用的“标准语言”，才谈得上跨技术平台的应用集成。

EAI 是 Web Service 技术出现的结果，Web Service 将传统的“服务提供者—服务请求者”服务模式改进为“服务提供者—服务请求者—服务目录”三方服务模式，通过 SOAP、UDDI、WSDL 等一系列基于 XML 的协议和标准，定义了服务发现、服务请求和服务绑定三种机制，从而实现了应用资源的标准化、服务化和社会化。

EAI 也是设计模式中的软件架构模式的发展结果。所谓设计模式是设计方案的一种抽象，它反映了软件设计人员为了提高软件重用性和灵活性而得到的一种设计结构。架

构模式（Architecture Pattern）是专门处理涉及应用集成的复杂性的设计模式。与 EAI 相关的架构模式包括集成适配器（Integration Adapter）、集成消息器（Integration Messenger）、集成面（Integration Façade）、集成媒介器（Integration Mediator）等。有关设计模式的详细知识，将在本书的第 12 章进行介绍。

EAI 所依赖的技术很新、很复杂，进行 EAI 是一项琐碎而艰巨的工程，但并不是要一切从“造砖”开始。相反，EAI 的思想是标准化和集成，提倡用成熟的框架、平台和中间件等中间产品来支持 EAI。有许多软件产品支持这一过程，包括应用服务器产品、中间件产品、门户开发平台等，还有许多厂商推出了支持 EAI 的完全解决方案。

9.2 EAI 集成模型

EAI 的最终目标就是将整个企业的应用程序快速、方便地集成在一起，它代表了系统设计方法和技术的变革，其目的是减少现今系统集成工作的复杂程度。

集成模型是指一种用来集成软件的特定方法和结构，它定义了集成的特性和机制，并由此决定如何将软件集成在一起。集成模型的重点在于：

（1）实现集成的简单性。

（2）对于不同配置的集成，具有较好的复用性。

（3）有较广泛的集成方法可用。

（4）在执行集成时要求的专门技术。

最常见的三种集成模型是表示集成、数据集成和功能集成。表示层的集成最容易实现，但也有很大的局限性；数据集成则提供了更加广泛的解决方法，但是可能要重写软件才能正确地处理各种数据；而功能集成是最重要的模型，也是最复杂的模型。

根据在集成时，应用软件或数据库的内部构造是否可见，可以分成白盒集成和黑盒集成两种。白盒集成方法中，集成者了解应用程序或数据库的内部结构。而黑盒集成则表示对集成者隐藏了软件和数据库的内部特定，只是使用 API、连接器或接口来实现。

9.2.1 表示集成

表示集成模型就是想创建一个新的用户界面，使用现有的表示逻辑模块来访问原有应用软件，用户的每个操作都将映射到原有的系统中去。表示集成模型的原理如图 9-1 所示。

正如图 9-1 所示，在原有两个不同软件的基础上，建立一个公用表示界面，从而达到将两个应用集成在一起的目的。而且整个集成工作都采用的是黑盒集成方法，无须了解程序与数据库的内部构造。屏幕截取、输入模拟技术是在此模型中常用的集成技术。表示集成模型通常应用于以下几种情况：

（1）在现有的基于终端的应用软件上配置基于 PC 的用户界面。

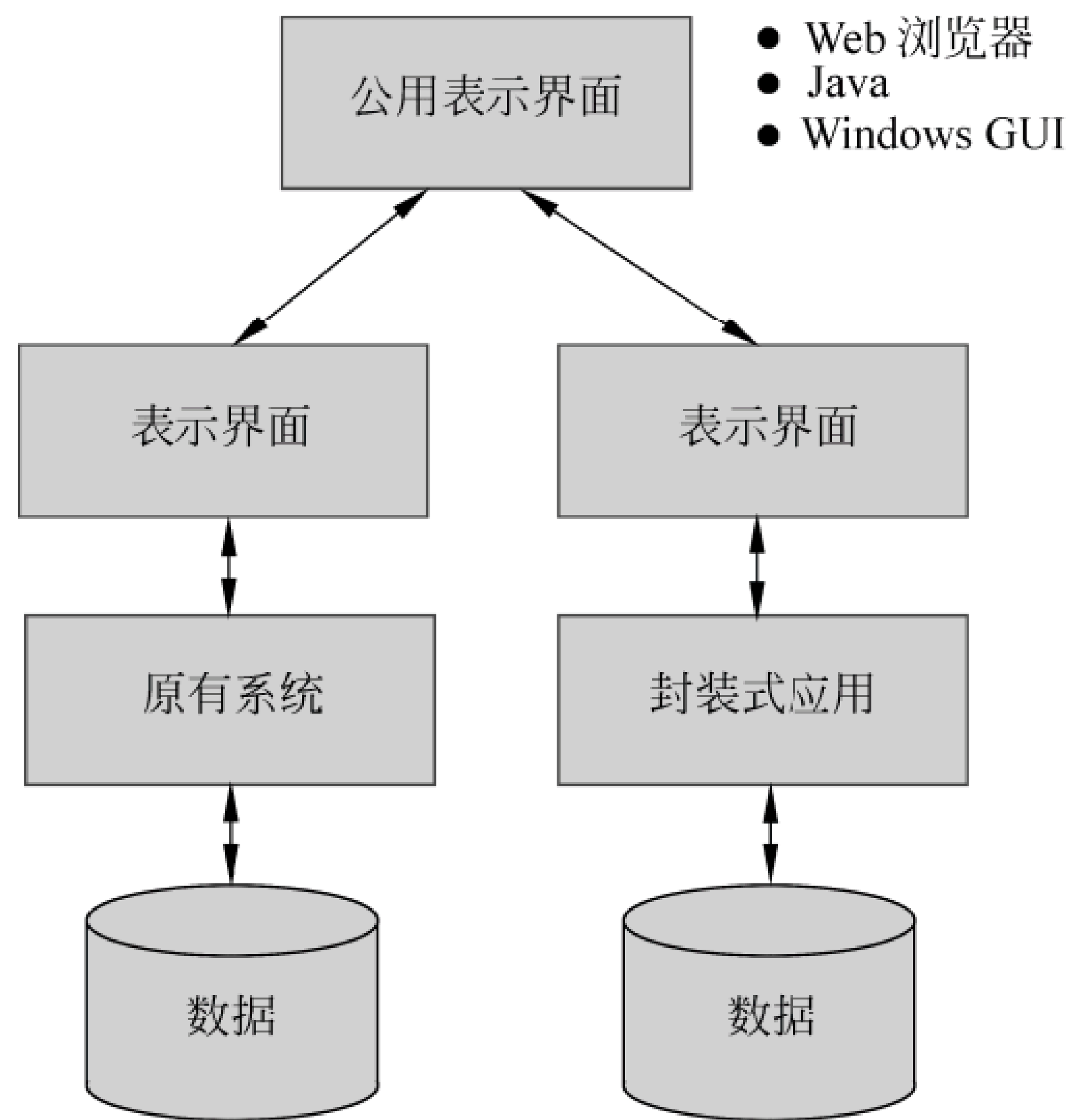


图 9-1　表示集成模型

（2）为用户提供一个看上去统一，但是由多个软件组成的应用系统。

（3）当只有可能在显示界面上实现集成时。

从上面的描述中，可以得知，表示集成的实现是很简单的，也是很不彻底的，只是做了一层“外装修”，而额外多出来的这块界面层也将可能成为系统的性能瓶颈。

9.2.2　数据集成

数据集成则是跳过界面和业务逻辑层，直接从应用软件的数据库或数据结构开始进行集成，正如图 9-2 所示的，数据集成就需要集成者对应用软件和数据库的内部构造有一定的了解。

有许多不同的中间件工具可以用于数据集成。例如，批量文件传输，即以特定的或是预定的方式在原有系统和新开发的应用软件之间进行文件传输；用于访问不同类型数据库系统的 ODBC 标准接口；向分布式数据库提供连接的数据库访问中间件技术。

通常在以下情况下，将会应用数据集成：

（1）需要对多种信息源产生的数据进行综合的分析和决策。

（2）要处理一些多个应用程序需要访问的公用信息库。

（3）当需要从另一个数据源获得数据来更新某个数据源时，特别是它们之间的数据格式不相同时。

相对而言，数据集成比表示集成要更加灵活，但是当业务逻辑经常发生变化时，数据集成就会面临困难。

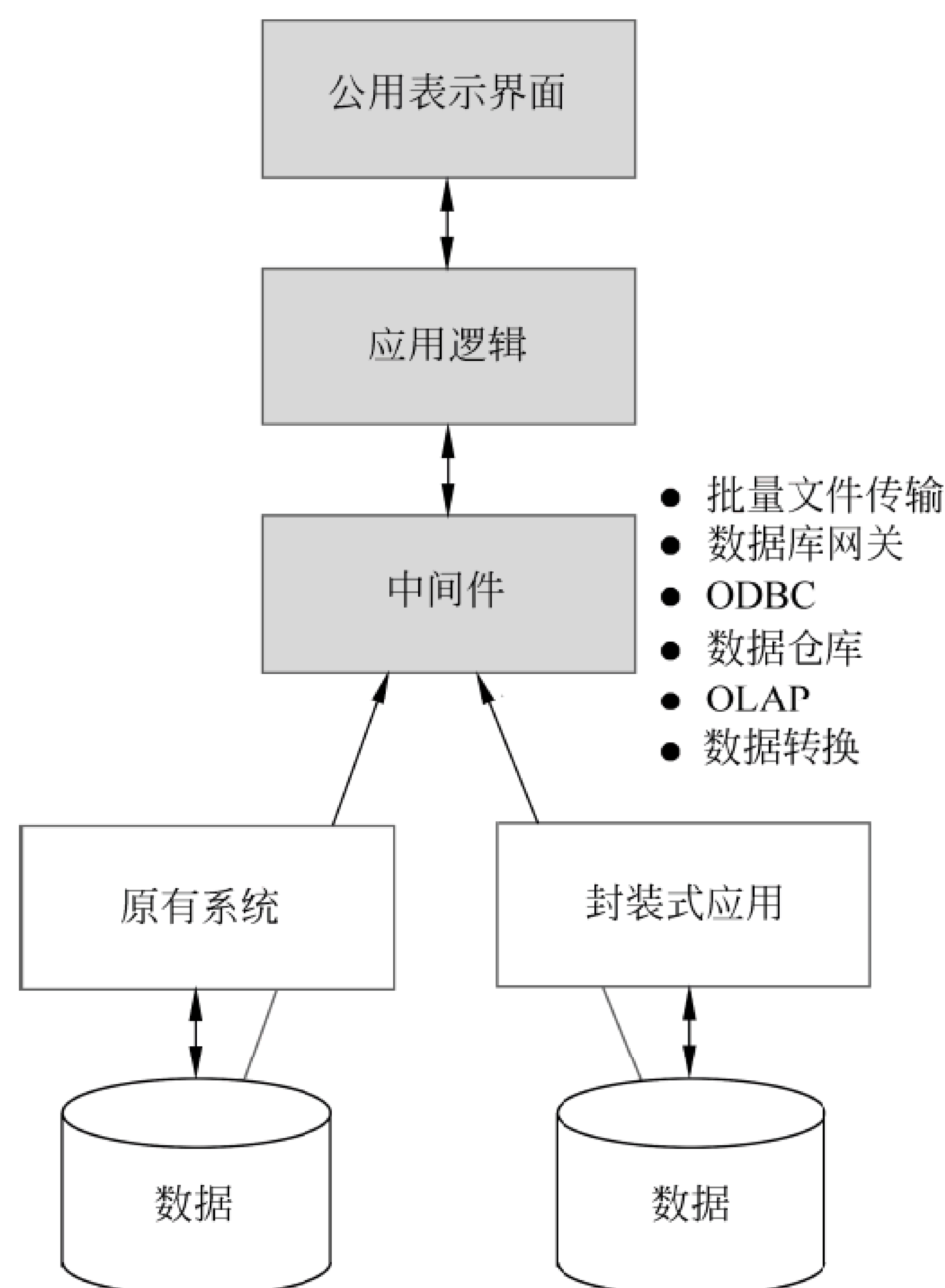


图 9-2 数据集成模型

9.2.3 功能集成

正如表示集成是在界面层进行集成，数据集成是在数据层进行集成一样，功能集成则是在业务逻辑层上进行集成的。功能集成的集成点存于程序代码中，集成处可能只需简单使用公开的 API 就可以访问，当然也可能需要添加附加的代码来实现。如图 9-3 所示，功能集成也是黑盒集成。

实现功能集成时，可以借助于以下几种方法。

（1）远程过程调用：早期广泛使用，但由于其只能够提供访问的定义和基本的通信能力，而且开发十分不便，因此随着分布式处理中间的发展，也就逐步不再使用了。

（2）面向消息的中间件：也就是通过在新旧应用软件、不同应用软件之间进行消息传递实现集成。

（3）分布式对象技术：如 CORBA、DCOM、.NET、J2EE 等。

（4）事务处理监控器：结合两阶段提交等技术来控制传输，以保障分布式事务的成功处理，例如，BEA 公司的 Tuxedo 等就属于这种方法。

功能集成与另外两种集成模型比较起来，灵活性更高，可以采用数据同步、多步处理以及即插即用构件的方法来实现。通常应用于以下场合：

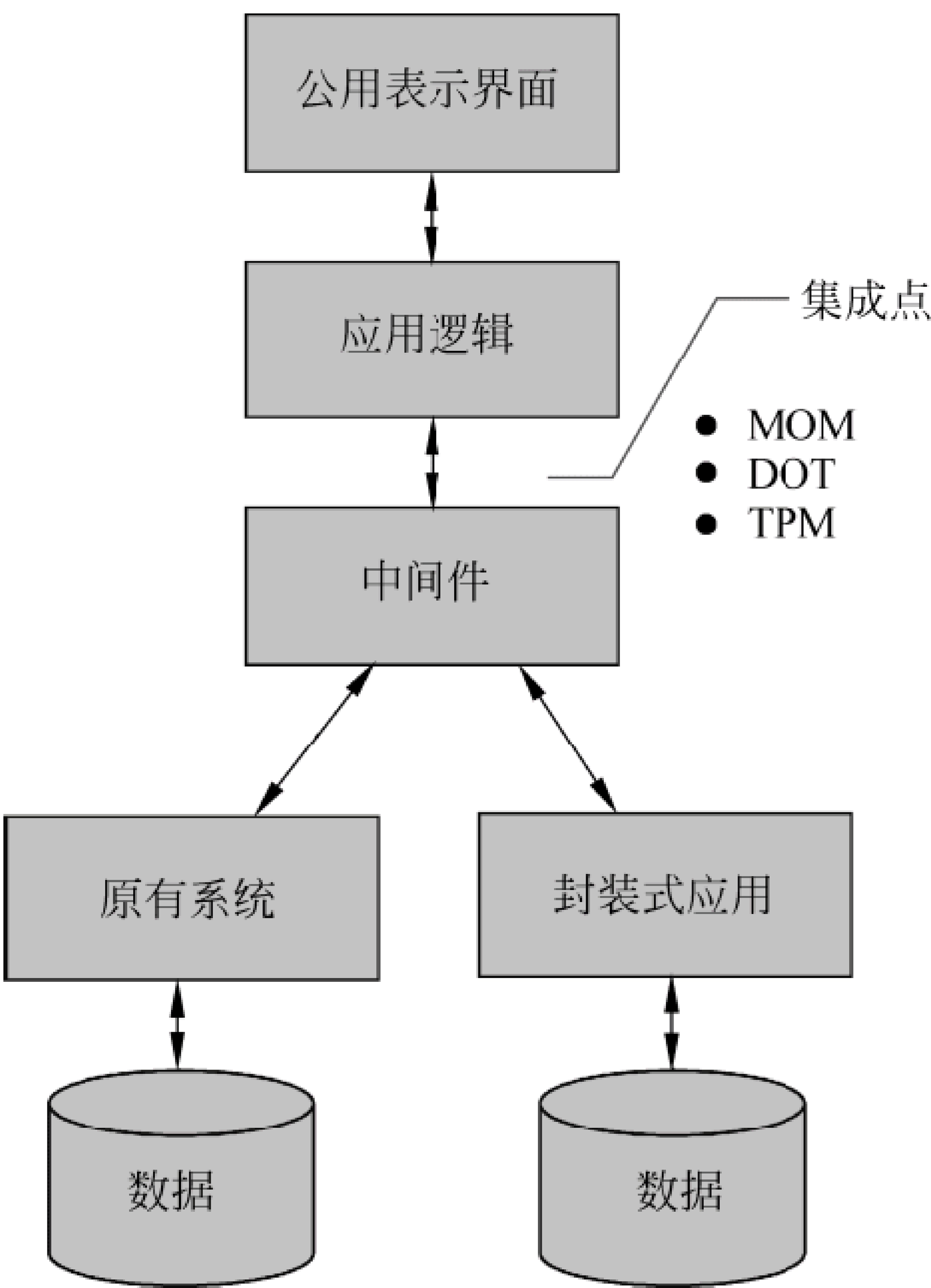


图 9-3　功能集成模型

（1）在现有的基于终端的应用软件上配置基于 PC 的用户界面。

（2）为用户提供一个看上去统一，但是由多个软件组成的应用软件。

（3）需要对多种信息源产生的数据进行综合的分析和决策。

（4）要处理一些多个应用程序需要访问的公用信息库。

（5）当需要从另一个数据源获得数据来更新某个数据源时，特别是它们之间的数据格式不相同时。

也就是说，表示集成和数据集成适用的环境下，都适用于功能集成模型。但是，由于功能集成是在业务逻辑层进行的，因此复杂度较之更高一些。而且还要注意的是，很多软件的业务逻辑部分并没有提供 API，这样，集成难度就会很大。

9.3　EAI 与标准化

EAI 是一种基于标准的集成，它所依赖的标准体系很复杂，需要用到多种类型的标准。从整个社会的层面上看，EAI 标准体系的建设非常重要。而其中应用标准，尤其是内容标准又值得特别重视。因为基础性和技术性标准的制定是相对容易见效，甚至是“一劳永逸”的事情，而内容标准的建设则是一项长期、艰巨、变化的任务。

缺乏内容标准的直接后果就是造成“信息孤岛”。这一点从 2003 年的 SARS 事件中可以领略到。从硬件基础上来说，由政府投资建设的“金卫”工程已经把不同地域、不

同级别、不同领域（医学研究、教学、医疗、疾病预防和控制、血液等）的所有卫生机构连接起来了，但人们还是被突如其来的 SARS 打了个措手不及：不仅不同领域的信息不能及时互通，就是同一领域、不同地区的信息也不能很好互通。结果是流派林立、孤岛纷呈，只能借助强大的行政力量和社会资源来弥补信息网络的软弱，公共卫生信息化的投入远远没有发挥应有的作用。究其原因，就是以前的建设重点多放在物理资源建设和特定功能的应用系统建设上，而没有对行业、领域信息化的内容进行标准化，以及没有基于标准化的一个自上而下的 EAI 工程。自从此次事件以后，有关人员看到了这个问题，也加强了在这方面的建设投入，如今，面貌得到了改观。

1．应用层标准

应用标准是在特定的信息化应用背景下，对信息内容、语义、信息处理逻辑的规定。它可以直接用于指导信息化应用的建设。从应用目的来区分，应用层标准又包括内容标准、技术标准和过程标准。

（1）内容标准：对特定领域或行业的信息内容进行的规定。例如，HL7（Health Level 7）就是 ANSI 的一个公共卫生领域的内容标准，它涵盖了医疗、保健、健康保险、疾病控制等公共卫生相关领域中对信息内容的规定。在美国进行公共卫生行业相关的信息化项目必须遵守 HL7 标准，这样就使得系统一经建成就可以实现互通，从宏观上节省了大量的社会资源；可共享内容对象参考模型（Sharable Content Object Reference Model，SCORM）是一个个人学习领域的内容标准，在美国被教育和培训机构广泛接受。任何一门课程，以及任何个人学习的状况都成为一个社会化的资源，可以随着学习者的迁移而流动。在英国的电子政府建设体系中，有一个称为 Government Category List（政府目录列表）的资源目录，也是一种内容标准。在内容标准方面，我国有关部门也在积极制订相关规范，但由于多方面的原因，实现起来比较困难。

（2）技术标准：技术标准是在特定的应用环境中对于信息化的一些技术方面的规定。例如，行业的编码标准、通信标准、安全标准、接口标准等，我国电子政务标准体系中的工程管理规范、信息安全规范和网络基础设施都属于此类。

（3）过程标准：对特定领域信息化建设过程方面的规定。例如，我国的信息工程监理规范就属于此类。

2．元标准

元标准也称为元数据标准，它不是一种可以直接指导信息化应用建设的标准，而是用于描述其他应用层标准的标准。

元标准的出现是为了适应标准建设本身的动态性。XML 的出现及相关技术的发展为实现标准化的动态性提供了技术平台。有了元标准，就可以以最小的代价演进应用层标准。因此，现在的几乎所有应用标准，尤其是内容标准，都在使用 XML 这个最基本的元标准。例如，电子商务标准从行政管理、商务与运输用电子资料交换（Electronic Data Interchange for Administration, Commerce, and Transport，EDIFACT）向 ebXML 的迁移就

是一个很好的例子，而上面提到的 HL7、SCORM 等，都是基于 XML 的应用标准。

元标准也是有层次的。XML 是所有元标准的基础，而例如我国电子政务标准体系中的主题词、XML 技术应用等规范，都是基于 XML 的、指导应用层标准建设，但又不属于应用标准的规范。

3．信息孤岛形成的原因

有一些普遍存在的错误观念是造成我国应用标准化滞后的原因。

（1）标准化制定和推进部门缺乏对应用标准的认识、研究深度和推进力度，在组织结构、基础设施上缺乏必要的投入。

（2）应用建设单位把标准化看做一个僵死的、静态的结果，而不是把标准看做有层次的结构，把标准化看做一个动态的过程；普遍存在一种对标准的观望态度。例如，在医院信息系统（Hospital Information System，HIS）领域曾经一度有一种流行看法，认为某些医学技术标准不太适合于计算机化表示，因而会严重影响 HIS 的建设进程。有许多医院和建设单位甚至因为观望类似于“性别编码”这个层次的“权威标准”而迟迟不敢进行建设。

（3）某些信息决策者认为 EAI 标准化就是要把数据、流程等一切掀翻重来，从顶向下，全部采用统一的技术平台、统一的处理流程，因而把标准化当做一项要么“绝对标准”，要么完全没有的工程，认为标准化的代价很大、周期很长，应该谨慎推行。

正是由于有这些观念，造成了“标准化太困难、代价太大→非标准的应用系统建设先上→形成新的信息孤岛→集成更加困难”的恶性循环现状。

EAI 的初衷就是要消除信息孤岛、实现整合。因此，在 EAI 中，尤其是在企业级、区域级、行业级和类似电子政务这种影响整个信息社会的“社会级”的 EAI 工程中，一定要注意标准，尤其是应用标准的建设（包括标准内容和标准平台建设）。这是 EAI 工程的信息主管部门和首席信息官（Chief Information Officer，CIO）的职责，也是任何一个技术提供者（包括商用技术平台提供商和非商业的技术联盟）都不会真正全心全意去做的事情。

9.4　EAI 的实施

EAI 的实施涉及到遗留系统的处置、新系统的定位，还要涉及业务流程的改变，是一个时间和空间跨度都很大的工程。没有一个完善的方法论不可能收到理想的效果。但由于不同的 EAI 项目的需求特点、实施基础是不同的，也不可能有一个包医百病的良方。作为一种集成项目，WebLogic 提出的 GEAR（Gather，Explore，Assemble，Roll-out）具有一般的指导意义。

（1）Gather：即收集需求，让企业去收集所有的需求，并规范需求。

（2）Explore：即探究，详细地分析和设计企业所有的需要，不管是信息或是流程方

面，都得去做探究。

（3）Assemble：即装配，所有解决方案的开发和测试都在这个阶段。

（4）Roll-out：即上线。

GEAR 跨越了 EAI 集成项目的整个生命周期，WebMethod 还为项目实施提供了众多的模版、工具和文档支持。

就像对待软件工程中的传统瀑布模型一样，也不必把 GEAR 看做 EAI 的一个僵死的步骤划分。完全可以根据项目需要，科学、合理地运用其他软件工程方法。例如，对于需求不是很明确，遇到的阻力比较大的 EAI 项目，可能在 Gather 和 Exlore 阶段形成循环，也许还要引用原型思想，对 EAI 带来的效果进行严谨的分析和直观展示；Assemble 和 Roll-out 也不必一次到位，可以结合战略目标合理地分期实现。

1．实施的原则

EAI 项目的实施可以是多样的，但有几个原则是不可忽略的：

（1）集成原则。EAI 的目标之一是整合信息资源，使组织的信息化建设过程走向有序化。如果在工程中产生新的信息孤岛，那就与设定的目标背道而驰了。

（2）标准原则。没有标准和统一架构的 EAI 将退化成过时的点到点集成，是没有意义的。

（3）管理配套原则。EAI 是一项以新的信息技术提高组织可管理性、满足组织战略需求的工程，不能脱离管理需求来做 EAI。

（4）拿来原则。EAI 是一项集成工程，需要不同层次、规模的工具支持。现在各种专用技术和市场都比较成熟，选择范围比较大。合理选用成熟技术和产品，可以保证项目质量，提高效率。

2．实施的阶段

在企业中，实施 EAI 通常需要经历局部试点、建立体系结构、推广体系结构和企业 EAI 化 4 个阶段。

（1）局部试点阶段。由于 EAI 技术涉及面广，牵扯的范围大，因此，在企业中实施 EAI 时，最好的办法是从一个要开发的新应用软件系统着手。在这个新应用软件系统的开发时应用 EAI 的技术和方法，以积累成功的经验，并且从中摸索出一个可以再次利用的、企业级的 EAI 体系结构雏形。

（2）建立体系结构阶段。当通过局部试点，摸索出符合企业情况的 EAI 体系结构雏形之后，就可以开始对这个体系结构进行完善。在本阶段中，应该成立一个专门的工作组，通过对原有系统的分析，以及在第一个阶段总结的经验的基础上，进一步建立完善的体系结构。

希赛教育专家提示：在本阶段中，体系结构的建立不是一个纯理论的行为，而是应该本着“边应用边总结”的策略进行，也就是要继续在新的应用软件中去应用，逐渐建立。

（3）推广体系结构阶段。当完成了企业级的 EAI 体系结构建立之后，就可以开始将其推广到整个企业中去。这时企业可以通过前两个阶段所积累的方法论、相关技术，更好地实施 EAI。本阶段的最终目标就是将 EAI 以结构化、规则化地进行应用。

（4）企业 EAI 化阶段。在这个阶段，EAI 已经融入了企业之中，其实施已经不成为问题，企业的精力放在了学习、调整、发展阶段中。这时的企业不仅能够有效地应用 EAI 体系结构、方法论和相关技术，而且还可以有效地改变、估算 EAI 的策略，灵活地应用。

本章参考文献

[1] 江泽锋. 基于 Web Services 的企业应用集成研究与应用. 重庆大学硕士学位论文，2004

[2] 张博，杨丽君. 企业应用集成. 北京：机械工业出版社，2003

[3] 黄双喜，范玉顺，赵大哲. 基于 Web 服务的企业应用集成. 计算机集成制造系统，2003

[4] 吴志刚，赵菁华. 我国电子政务标准化工作概况. 中国标准化，2003

第 10 章　可扩展标记语言

XML 是一套定义语义标记的规则，这些标记将文档分成许多部件并对这些部件加以标识。它也是元标记语言，用于定义其他与特定领域有关的、语义的、结构化的标记语言的句法语言。XML 与 HTML 一样，都来自标准通用标记语言（Standard Generalized Markup Language，SGML）。

10.1　XML 概述

SGML 是一种用标记来描述文档资料的通用语言，它包含了一系列的文档类型定义（Document Type Definition，DTD），DTD 中定义了标记的含义，因而 SGML 的语法是可以扩展的。SGML 十分庞大，既不容易学，又不容易使用，在计算机上实现也十分困难。鉴于这些因素，Web 的发明者（欧洲核子物理研究中心的研究人员）根据当时（1989 年）计算机技术的能力，提出了 HTML 语言。

HTML 只使用 SGML 中很小一部分标记，例如，HTML 3.2 定义了 70 种标记。为了便于在计算机上实现，HTML 规定的标记是固定的，即 HTML 语法是不可扩展的，它不需要包含 DTD。HTML 这种固定的语法使它易学易用，在计算机上开发 HTML 的浏览器也十分容易。正是由于 HTML 的简单性，使 Web 技术从计算机界走向全社会，走向千家万户，Web 的发展如日中天。

随着 Web 的应用越来越广泛和深入，人们渐渐觉得 HTML 不够用了，HTML 过于简单的语法严重地阻碍了用它来表现复杂的形式。尽管 HTML 推出了一个又一个新版本，已经有了脚本、表格、帧等表达功能，但始终满足不了不断增长的需求。另一方面，计算机技术的发展也十分迅速，已经可以实现比当初发明 HTML 时复杂得多的 Web 浏览器，所以开发一种新的 Web 页面语言既是必要的，也是可能的。

有人建议直接使用 SGML 作为 Web 语言，这固然能解决 HTML 遇到的困难。但是 SGML 太庞大了，用户学习和使用不方便尚且不说，要全面实现 SGML 的浏览器就非常困难。

于是，Web 标准化组织 W3C 就提出了简化 SGML、提供超链接能力、样式表、使用简单、文件解析等几个改进方案，并实现以下 10 个目标。

- 能够直接应用在 Internet 上：克服 SGML 没有支持 Web 的通信协议，因此，需要对其改进，使其支持 HTPP、URL、Script、Java 等，而且要尽量瘦身。
- 能被各式应用软件使用：可通过 XML 解析器来使用 XML 文件。

- 能与 SGML 兼容：SGML 已经使用多年，因此，保持兼容性能够最有效地利用原有资料。
- 能轻易发展 XML 相关软件：也就是需要有效降低开发 XML 应用软件的成本。
- 能简化 SGML：SGML 具有大量不常使用的可选项，XML 应尽可能减少类似情况。
- XML 文件可读性高：也就是 XML 文件应该具有良好的结构性。
- XML 规范能尽完成：HTML 的瓶颈效应越来越明显，因此，应该尽可能地快速提出 XML。
- XML 规范必须简洁：也就是降低学习时间和成本。
- XML 文件易于建立：允许 XML 文件中不一定带有 DTD 文档。
- 语法不可模糊不清：克服 HTML 中的相关问题。

10.1.1　XML 的特点

根据 W3C 提出的方案，XML 语言应运而生，与传统的 SGML 或 HTML 相比，XML 具有以下特点。

1．简洁有效

XML 是一个精简的 SGML，它将 SGML 的丰富功能与 HTML 的易用性结合到 Web 应用中。XML 保留了 SGML 的可扩展功能，这使 XML 从根本上有别于 HTML。XML 要比 HTML 强大得多，它不再是固定的标记，而是允许定义数量不限的标记来描述文档中的资料，允许嵌套的信息结构。HTML 只是 Web 显示数据的通用方法，而 XML 提供了一个直接处理 Web 数据的通用方法。HTML 着重描述 Web 页面的显示格式，而 XML 着重描述的是 Web 页面的内容。

XML 中包括可扩展样式语言（Extensible Style Language，XSL）和可扩展链接语言（Extensible Linking Language，XLL）。XSL 用于将 XML 数据翻译为 HTML 或其他格式的语言。XSL 提供了一种叠式页面 CSS（Cascading Style Sheets）的功能，使开发者构造出具有表达层结构的 Web 页面，以有别于 XML 的数据结构。XSL 也能和 HTML 一起构造叠式页面。XSL 可以解释数量不限的标记，它使 Web 的版面更丰富多彩，例如，动态的文本、跑马式的文字等。此外，XSL 还处理多国文字、双字节的汉字显示、网格的各种各样的处理等；XLL 是 XML 的链接语言，它与 HTML 的链接相似，但功能更强大。XLL 支持可扩展的链接和多方向的链接，它打破了 HTML 只支持超级文本概念下最简单的链接限制，能支持独立于地址的域名、双向链路、环路、多个源的集合链接等。XLL 链接可不受文档制约，完全按用户要求来指定和管理。

2．易学易用

为了使 XML 易学易用，XML 精简了一大片 SGML 难得用一次的功能。SGML 中常用的部分只占 20%，XML 抛弃了 SGML 中不常用的部分，使它一下就精简了 80%。

这样一来，XML 的语法说明书只有 30 页，而 SGML 的语法说明书却有 500 页。

XML 设计中也考虑了它的易用性，易用性来自两个方面：一方面用户编写 Web 页面方便，另一方面设计人员实现 XML 浏览器也不太困难。

3．开放的国际化标准

XML 是 SGML 在市场上有许多成熟的软件可用来帮助编写、管理等，XML 通过验证的标准技术，使得其具有高度的开放性。众多业界顶尖公司，与 W3C 的工作组并肩合作，协助确保交互作业性，支持各种系统和浏览器上的开发人员、作者和使用者，以及改进 XML 标准。其中包括：

（1）XML 标准。这是 W3C 正式批准的，这意味着这个标准是稳定的，完全可用于 Web 和工具的开发。

（2）XML 名域标准。用来描述名域的句法，支持能识别名域的 XML 解析器。

（3）文档对象模型（Document Object Model，DOM）标准。为给结构化的数据编写脚本提供标准，这样，开发人员就能够与计算机在基于 XML 的数据上进行交互。

（4）XSL 标准。XSL 有两个模块：XSL 转换语言和 XSL 格式化对象。其中转换语言可用来转换 XML 以满足显示要求。由于 XSL 的两部分是模块化的，因此，转换语言能够独立地用来进行多用途的转换，包括把 XML 转换成结构完整的 HTML。

（5）XLL 标准和 XML 指针语言（XPointer）标准。XLL 提供类似与 HTML 的链接，但功能更强大。例如，链接可以是多方向的，可以存在于对象上而不仅仅是页面上。

希赛教育专家提示：XML 通过采用一个新的编码标准，可以支持世界上大多数文字。因此，XML 不仅能在不同的计算机系统之间交换信息，而且能跨国界和超越不同文化疆界交换信息。

4．高效且可扩充

XML 支持复用文档片断，使用者可以发明和使用自己的标签，也可与他人共享，可延伸性大。在 XML 中，可以定义无限量的一组标注。XML 提供了一个表示结构化资料的架构。一个 XML 构件可以宣告与其相关的资料为零售价、营业税、书名、数量或其他任何数据元素。随着世界范围内的许多机构逐渐采用 XML 标准，将会有更多的相关功能出现。XML 提供了一个独立的运用程序的方法来共享数据，使用 DTD，不同组中的人就能够使用共同的 DTD 来交换数据。应用程序可以使用 DTD 来验证用户接收到的数据是否有效，也可以使用一个 DTD 来验证用户自己的数据。

总之，XML 使用一个简单而又灵活的标准格式，为基于 Web 的应用提供了一个描述数据和交换数据的有效手段。HTML 描述了显示全球数据的通用方法，而 XML 提供了直接处理全球数据的通用方法。

10.1.2 XML 的作用

XML 语言可以让信息提供者根据需要，自行定义标记及属性名，结构化地描述信息内容，因此赋予了应用软件强大的灵活性，为开发者和用户带来许多好处。

（1）使得搜索更加有意义。在非 XML 构建的 Web 系统中，搜索软件必须了解每个数据库是如何构建的。这实际上是不可能的，因为每个数据库描述数据都是不同的。而 XML 是一个自描述的语言，它可以使得数据“知道”自身的信息，从而可以进行更有使用价值的搜索功能。

（2）开发灵活的 Web 应用软件。XML 的应用将使得三层 Web 应用软件的开发更加简单，由于其能够有效地进行消息与数据标准的统一，从而设计、开发出更加灵活的 Web 应用软件。

（3）实现不同数据的集成。不同的数据库系统，其存储结构、应用程序接口都存在着许多不同点，因此基本上无法开发出一套能够针对这些相互不兼容的数据库的查询程序。而 XML 的出现，则改变了这个现象，由于数据是结构化的，因此，即使它们的来源不同，也是能够很容易地结合在一起。在开发时，可以在中间层的服务器上对从后端数据库和其他应用系统来的数据进行集成。然后，数据就能被发送到客户或其他服务器做进一步的集合、处理和分发。

（4）使用于多种应用环境。XML 的高扩展性、高灵活性特性，使得其可以描述各种不同种类的应用软件中的各种不同类型的数据。另外，XML 具有自描述性，可以很容易进行交换、处理，而且还不需要多余的内部描述。

（5）客户端数据处理与计算。由于 XML 格式的标准化，许多浏览器软件都能够提供很好的支持。因此，只需要简单地将 XML 格式的数据发送给客户端，客户端就可以自行对其进行编辑和处理，而不仅是显示。而且，XML 的 DOM 还允许客户端利用脚本或其他编程语言处理数据，而无需回到服务器端。这种将数据视图与内容分离的机制，可以更容易地创建出基于 Web 的、功能强大的应用，而无须基于高端数据库。

（6）数据显示多样化。XML 将显示和数据内容分离，提供了一种简单、开放、扩展的方式来描述结构化数据。与 HTML 不同的是，HTML 描述了数据的外观，而 XML 则描述的是数据本身。因此，XML 定义的数据可以指定不同的显示方式，利用 CSS 或 XSL 等工具来提供显示机制。

（7）局部数据更新。通过 XML，数据可以实现局部的更新。也就是说，当有其中的一部分数据变化时，并不需要重发整个结构化的数据，服务器只需将变化的元素发送给客户。而不是只要一条数据变化了，整个页面都必须重建。而且，还可以将新增加的信息加入到已存在的页面中，这样，就可以使得应用具有更高的性能。

（8）与现有 Web 发布机制相兼容。由于 XML 也是一个开放的基于文本的格式，可以像 HTML 一样使用 HTTP 进行传送，不需要对现存的网络进行变化，可以很方便地应用于 Web 上的数据发布。

（9）可升级性。由于 XML 彻底把标识的概念与显示分开，处理者能够在结构化的数据中嵌套程序化的描述以表明如何显示数据。这是一个强大的机制，使得客户计算机

与使用者间的交互作用尽可能地减少了。同时，也减少了服务器的数据交换量和浏览器的响应时间。另外，XML 减少了服务器的工作量，大大增强了服务器的升级性能。

（10）压缩性能高。XML 压缩性能很好，因为用于描述数据结构的标签可以重复使用。XML 数据是否要压缩要根据应用来定，还取决于服务器与客户间数据的传递量。XML 能够使用 HTTP 1.1 中的压缩标准。

10.1.3 XML 的应用

XML 的种种优点可以在以下几种应用情况下发挥其巨大的作用：

（1）应用于客户需要与不同的数据源进行交互时。数据可能来自不同的数据库，它们都有各自不同的复杂格式。但客户与这些数据库间只通过一种标准语言进行交互，那就是 XML。由于 XML 的自定义性及可扩展性，它足以表达各种类型的数据。客户收到数据后可以进行处理，也可以在不同数据库间进行传递。也就是说，在这种情况下，XML 解决了数据的统一接口问题。

（2）应用于将大量运算负荷分布在客户端。客户可根据自己的需求选择和制作不同的应用程序以处理数据，而服务器只须发出同一个 XML 文件。如按传统的客户/服务器工作方式，客户向服务器发出不同的请求，服务器分别予以响应，这不仅加重服务器本身的负荷，而且网络管理者还须事先调查各种不同的用户需求以做出相应不同的程序，但假如用户的需求繁杂而多变，则仍然将所有业务逻辑集中在服务器端是不合适的，因为服务器端的编程人员可能来不及满足众多的应用需求，也来不及跟上需求的变化，双方都很被动。应用 XML 则将处理数据的主动权交给了客户，服务器所作的只是尽可能完善、准确地将数据封装进 XML 文件中，正是各取所需、各司其职。XML 的自解释性使客户端在收到数据的同时也能理解数据的逻辑结构与含义，从而使广泛、通用的分布式计算成为可能。

（3）应用于将同一数据以不同的面貌展现给不现的用户。XML 类似于同一个剧本，却可以用电视剧、电影、话剧、动画片等不同形式表现出来。这一应用将会为网络用户界面个性化、风格化的发展铺平道路。

（4）应用于网络代理对所取得的信息进行编辑、增减以适应个人用户的需要。有些客户取得数据并不是为了直接使用而是为了根据需要组织自己的数据库。例如，希赛教育网站上建立了一个庞大的题库，实际考试时，随机将题库中的若干题目取出组成试卷，再将试卷封装进 XML 文件。接下来便是最精彩的部分，在各个考点让其通过一个过滤器，滤掉所有的答案，再发送到考生面前，未经过滤的内容则可直接送到老师手中。此外，XML 文件中还可以包含诸如难度系数、往年错误率等其他相关信息，这样，只需几个小程序和一个 XML 文件，便可变成多个文件传送到不同的用户手中。

10.2　解析 XML

XML 使用的是非常简单的数据格式，可以用 100%的纯美国信息交换标准代码（American Standard Code for Information Interchange，ASCII）文本来书写，也可以用几种其他定义好的格式来书写。ASCII 文本是几乎不会“磨损”的。丢失一些字节甚至是相当多的字节，剩下的数据还是可以读取的。这就与许多格式形成了鲜明的对比，例如，压缩数据或是串行的 Java 对象，这些数据即使丢失一个字节，剩余的数据也变得不可读取了。

更重要的是，XML 是自描述的。假设在 25 世纪有一个信息考古学者，他在某个地方发现了如下一大段经过时间的“冲刷”而保存下来的 XML 代码：

```
<PERSON ID="p1100" SEX="M">
      <NAME>
         <GIVEN>Judson</GIVEN>
         <SURNAME> McDaniel</SURNAME>
      </NAME>
      <BIRTH>
         <DATE>2 Feb 1934</DATE>
      </BIRTH>
      <DEATH>
         <DATE>9 Dec 2005</DATE>
      </DEATH>
</PERSON>
```

即使这个考古学家不熟悉 XML，但假设他可以讲 21 世纪的英语，那么就可以很好地了解名为 Judson McDaniel 的人，此人出生在 1934 年 2 月 21 日，而死于 2005 年 12 月 9 日。事实上，数据中有一些空白或是损坏，还是可以得到这些信息。但对于专有格式的电子表格或是字处理程序的格式，就不是这么回事了。

更进一步说，XML 有很好的规格文档。W3C 的 XML 1.0 规范和大量的论文书籍，如本书，都向人们准确地说明如何来阅读 XML 数据，没有什么秘密使得人们发生失误。

10.2.1　XML 与 HTML 的区别

虽然 XML 与 HTML 都是标记语言，但它们在结构和应用上有很大的区别。

HTML 是一种格式化的语言，一个 HTML 文本可以看作一个格式化的程序。HTML（及类似的）语言定义了一套固定的标记，用来描述一定数目的元素。如果标记语言中没有所需的标记，用户也就没有办法了。这时只好等待标记语言的下一个版本，希望在新

版本中能够包括所需的标记。另外，用 HTML 语言描述的程序或文本具有“内容＋格式”的双重属性。一个 HTML 程序在不同平台、不同浏览器上的表现是一模一样的。而一段符合 XML 语法规范的文本则是一段“纯”数据，它的结构由其他的称为 DTD 的文本来描述，而它的处理则可能是任何其他支持 XML 的容器或程序，例如，IE 浏览器依据相关的 CSS 或 XSL 文件来显示 XML 数据。开发人员可以用任何支持 XML 的开发工具开发自己的 XML 处理程序。

与 HTML 相比的另一个不同点是，XML 是一种元标记语言。它可以被用于定义其他的标记语言，甚至 DTD 和 XSL 文档也是用 XML 语法描述的。例如，在 Peter Murray-Rust 的化学标记语言（Chemical Markup Language，CML）中的 MOL.DTD 文件中描述了词汇表和分子科学的句法，其中包括 chemistry（化学）、crystallography（结晶学）、solid state physics（固体物理）等词汇。它包括用于 atoms（原子）、molecules（分子）、bonds（化学键）、spectra（光谱）等的标记。这个 DTD 可与分子科学领域中的许多不同的人共享。对于其他领域也有类似的 DTD，用户还可以创建自己的 DTD。

XML 定义了一套元句法，与特定领域有关的标记语言（如 MusicML、MathML 和 CML 等）都必须遵守。如果一个应用程序可以理解这一元句法，那么它也就自动地能够理解所有的由此元语言建立起来的语言。浏览器不必事先了解多种不同的标记语言使用的每个标记。事实是，浏览器在读入文档或是它的 DTD 时才了解给定文档使用的标记。

有了 XML，就意味着不必等待浏览器的开发商来满足用户的需要了。用户可以创建自己需要的标记，当需要时，告诉浏览器如何显示这些标记就可以了。

10.2.2 XML 文档

一个实用的 XML 文档必须满足两点，分别是组织良好（Well-formed）和有效（valid）。下面，分别举例说明。

下例试图描述一个欢迎词：

```
<?xml version="1.0"?>
<visit>
        <to>alluser</to>
        <from>educity.cn</from>
        <heading>welcome</heading>
        <body>Welcome to the best online education website!</body>
</visit>
```

这就是一份可以接受的 XML 文档。由于使用了一些人们易于理解的标记如<note>、<body>等，觉得这些数据十分易读和有意义。

这也是一个组织良好的 XML 文档。即它满足以下三项基本规则：

（1）文档以 XML 定义<?xml version="1.0"?>开始。

（2）有一个包含所有其他内容的根元素，如上面例子中的<visit>和</visit>标记符。

（3）所有元素必须合理地嵌套，不允许交叉嵌套。

组织良好的 XML 可以对应为一棵逻辑上的树。没有组织好的文档在 HTML 中可能不算什么，因为浏览器已经被设计成可以处理这种问题。但是，在 XML 中却是致命的，因为应用程序将拒绝处理没有组织好的文件。

XML 文档单是组织良好是不够的。例如，如下的“欢迎词”数据：

```
<?xml version="1.0"?>
<visit>
        <to>alluser</to>
        <from>educity.cn</from>
        <heading>Welcome</heading>
</visit>
```

这段数据是组织良好的，但却不会产生什么实用价值。因为它没有关键的内容（<body>）。因此，还必须要有一个文档对 XML 数据文档的内容作出规定，这种文档就是 DTD。例如，关于 visit 的 DTD 可以这么写：

```
<?xml version="1.0"?>
<!DOCTYPE visit [
        <!ELEMENT visit (to,from,heading,body)>
        <!ELEMENT to (#PCDATA)>
        <!ELEMENT from (#PCDATA)>
        <!ELEMENT heading (#PCDATA)>
        <!ELEMENT body (#PCDATA)>
]>
```

!DOCTYPE visit 第二行的节点是 XML 文档中的“visit”类型；第三行说明元素“visit”有 4 个元素“to，from，heading，body”；第四行定义了“to”元素的类型为“#PCDATA”；第五行定义了“from”元素的类型为“#PCDATA”，等等。

DTD 可以看做对 XML 数据的语法结构的描述。使用 DTD，不同的人就能够使用共同的 DTD 来交换数据。有专门的小程序对 XML 文档是否组织良好，以及是否符合 DTD 定义的有效性进行扫描、判断和报告。

除 DTD 外，还有一种方式来定义有效的 XML 文档，那就是 W3C 与 XML 标准一起定义的 XML 模式（XML Schema）。与 DTD 相比，XML 模式有几个优势：

（1）XML 模式使用 XML 语法。换句话说，XML 模式是一个 XML 文档。这意味着可以像处理任何其他文档一样处理模式。例如，可以编写一个用于转换的可扩展样式表语言（Extensible Stylesheet Language for Transformation，XSLT）样式表，该样式表将 XML 模式转换成具有自动生成的 JavaScript 代码的 Web 表单，其中的 JavaScript 代码可

以验证输入的数据。

（2）XML 模式支持数据类型。尽管 DTD 确实支持数据类型，但很明显，这些数据类型是从发布的角度开发的。XML 模式支持 DTD 中的所有原始数据类型（诸如标识和标识引用之类的类型）。它们还支持整数、浮点数、日期、时间、字符串、URL 和其他对数据处理和验证有用的数据类型。

（3）XML 模式是可扩展的。除了 XML 模式规范中定义的数据类型以外，还可以创建自己的数据类型，并且可以基于其他数据类型派生出新的数据类型。

（4）XML 模式有更强的表达能力。例如，下面例子是关于邮政地址的 DTD 描述相匹配的 XML 模式。在 XML 模式定义中，它增加了两个约束：<state>元素的值必须刚好是两个字符长，<postal-code> 元素的值必须与正则表达式 [0-9]{5}(-[0-9]{4})? 相匹配。用 DTD 无法做这些事，尽管这个模式比 DTD 长很多，但它更清楚地表达了有效的文档看起来是什么样子。

```
<!-- address.dtd -->
<!ELEMENT address (name, street, city, state, postal-code)>
<!ELEMENT name (title? first-name, last-name)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT last-name (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT postal-code (#PCDATA)>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="address">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="name"/>
        <xsd:element ref="street"/>
        <xsd:element ref="city"/>
        <xsd:element ref="state"/>
        <xsd:element ref="postal-code"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="name">
```

```
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="title" minOccurs="0"/>
        <xsd:element ref="first-Name"/>
        <xsd:element ref="last-Name"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="title"      type="xsd:string"/>
  <xsd:element name="first-Name" type="xsd:string"/>
  <xsd:element name="last-Name"  type="xsd:string"/>
  <xsd:element name="street"     type="xsd:string"/>
  <xsd:element name="city"       type="xsd:string"/>

  <xsd:element name="state">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:length value="2"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>

  <xsd:element name="postal-code">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
</xsd:schema>
```

10.2.3　CSS 与 XSL

XML 文档最终是要通过一定的应用程序来表现的，如浏览器。用浏览器来表现 XML 就要经过 CSS 或 XSL 这一媒介，它们都是样式语言，描述了 XML 数据与 HTML 标记的映射关系。或可以说，XSL 是对 XML 文档进行排版的语言。

1．CSS

CSS 之于 HTML 文档的作用类似于 MS Word 中的“样式”的作用，可以在某种程度上把 HTML 文档中的排版格式信息与其他数据信息分离开。

一个 CSS 样式单就是一组规则（rule），样式再根据特定的一套规则级联起来。每个规则给出此规则所适用的元素的名称，以及此规则要应用于哪些元素的样式。例如，下列清单是一首诗的 CSS 样式单：

```
POEM { display: block }
TITLE { display: block; font-size: 16pt; font-weight: bold }
POET { display: block; margin-bottom: 10px }
STANZA { display: block; margin-bottom: 10px }
VERSE { display: block }
```

此样式单有 5 个规则。每个规则有一个选择符（规则所应用的元素的名称）和一组适用于此元素实例的属性。第一个规则说明 POEM 元素应以块的形式（display：block）显示其内容。第二个规则说明 TITLE 元素应以 16 磅（font-size：16pt）、粗体（font-weight：bold）将其内容显示在块中（display：block）。第三个规则说明 POET 元素应通过自身显示在块中（display：block），并且与紧随其后的下一块相距 10 个像素（margin-bottom：10px）。第四个规则与第三个相同，所不同的只是前者应用于 STANZA 元素。最后，第五个规则只简单地说明 VERSE 元素也是显示在自己的块中。

定义了如上的 CSS 以后，HTML 文档就可以用“POEM”、“TITLE”来说明真正的数据了。

CSS 是专为 HTML 设计的，HTML 的标志必须和 CSS 标志兼容。例如，要正确地支持 CSS 的 nowrap 属性就要求废除 HTML 中非标准的但又是经常使用的 NOWRAP 元素。而 XML 元素没有任何预定义的格式规定，它甚至不限于 HTML，所以，利用 XML 可以开发更强大的方式来解决 HTML 的样式与内容分离的问题，那就是 XSL。虽然也可以用 XML 来写 CSS，但是已经很少有人那样做了。

2．XSL

XSL 是专门用于 XML 文档的样式单语言，可以把 XSL 当成一种能够把 XML 转变成 HTML 的语言，一种能够筛选和排序 XML 文档中数据的语言，一种能够根据 XML 的数据数值格式化 XML 数据的语言（例如，把负数显示成红色）。XSL 文档本身就是结构完整的 XML 文档。

以下是一个应用 XSL 的例子。假设有一个 XML 数据文档：

```
portfolio.xml
<?xml version="1.0"?>
<portfolio>
  <stock exchange="nyse">
     <name>zacx corp</name>
  <symbol>ZCXM</symbol>
  <price>28.875</price>
```

```
  </stock>
  <stock exchange="nasdaq">
      <name>zaffymat inc</name>
      <symbol>ZFFX</symbol>
      <price>92.250</price>
  </stock>
</portfolio>
```

为了把这个 XML 文档在浏览器中显示，使用如下 XSL 文件：

```
portfolio.xsl
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="2" bgcolor="yellow">
  <tr>
    <th>Symbol</th>
    <th>Name</th>
    <th>Price</th>
  </tr>
  <xsl:for-each select="portfolio/stock">
  <tr>
        <td><xsl:value-of select="symbol"/></td>
    <td><xsl:value-of select="name"/></td>
    <td><xsl:value-of select="price"/></td>
   </tr>
  </xsl:for-each>
  </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

最后，在原来的 XML 文档第二行前中加入对 XSL 的引用：

```
<?xml-stylesheet type="text/xsl" href="portfolio.xsl"?>
```

就可以实现在一个 HTML 表格中显示股票交易数据了。

3．CSS 与 XSL 比较

CSS 只能改变特定元素的格式，也只能以元素为基础。但 XSL 样式单可以重新排列

元素并对元素进行重排序。这种样式单可以隐藏一些元素而显示另外一些元素。更进一步说，还可以选择应用样式的标记，而不仅是基于标记的，而且，XSL 还基于标记的内容和特性，基于标记在文档中相对于其他元素的位置，以及基于各种其他的准则。

CSS 的优越性在于具有广泛的浏览器支持。但是 XSL 更为灵活和强大，可更好地适用于 XML 文档。而且，带 XSL 样式单的 XML 文档可以很容易地转换为带 CSS 样式单的 HTML 文档。

如果只是要对一些固定数据进行排版，可以使用“HTML＋CSS”方式；如果这些数据是与某些应用程序相关，并且独立于程序存在的，并且要独立于程序来使用，则应该充分使用 XML 技术，采用“HTML＋XML＋XSL”。

10.3 XML 编程接口

本节将研究 XML 的多种编程接口，这些接口为开发人员使用 XML 文档提供了一致的接口。

10.3.1 API 接口

有许多 API 可以使用，本节介绍最流行和广泛使用的 API 中的 4 种：DOM、用于 XML 的简单 API（Simple API for XML，SAX）、JDOM 和用于 XML 解析的 Java API（Java API for XML Parsing，JAXP）。

1．DOM

文档对象模型为 XML 文档的已解析版本定义了一组接口。解析器读入整个文档，然后构建一个驻留内存的树结构，然后代码就可以使用 DOM 接口来操作这个树结构。用户可以遍历树以了解原始文档包含了什么，可以删除树的几个部分，还可以重新排列树和添加新的分支，等等。

DOM 提供了一组丰富的功能，用户可以用这些功能来解释和操作 XML 文档，但使用它们是有代价的。在开发用于 XML 文档的原始 DOM 时，很多人也提出了 DOM 的几个问题：

（1）DOM 构建整个文档驻留内存的树。如果文档很大，就会要求有极大的内存。

（2）DOM 创建表示原始文档中每个东西的对象，包括元素、文本、属性和空格。如果用户只需关注原始文档的一小部分，那么创建那些永远不被使用的对象是极其浪费的。

（3）DOM 解析器必须在代码取得控制权之前读取整个文档。对于非常大的文档，这会引起显著的延迟。

这些仅仅是由 DOM 的设计引起的问题，撇开这些问题，DOM API 是解析 XML 文档非常有用的方法。

2．SAX

为了解决 DOM 问题，就有了 SAX 接口的产生。SAX 的几个特征解决了 DOM 的问题：

（1）SAX 解析器向代码发送事件。当解析器发现元素开始、元素结束、文本、文档的开始或结束时，它会告诉用户。用户可以决定什么事件对自己重要，而且可以决定要创建什么类型的数据结构以保存来自这些事件的数据。如果没有显式地保存来自某个事件的数据，它就被丢弃。

（2）SAX 解析器根本不创建任何对象，它只是将事件传递给应用程序。如果希望基于哪些事件创建对象，这将由编程者自己来完成。

（3）SAX 解析器在解析开始的时候就开始发送事件。当解析器发现文档开始、元素开始和文本时，代码会收到一个事件。应用程序可以立即开始生成结果；不必一直等到整个文档被解析完毕。更妙的是，如果只查找文档中某些内容，代码一旦找到所要找的东西就可以抛出一个异常。该异常会停止 SAX 解析器，然后代码用它找到的数据做它需要做的任何事。

SAX 解析器也有些问题引人关注：

（1）SAX 事件是无状态的。当 SAX 解析器在 XML 文档中发现文本时，它就向代码发送一个事件。该事件仅仅给用户发现的文本，它不告诉用户什么元素包含那个文本。如果想知道这一点，则用户必须自己编写状态管理代码。

（2）SAX 事件不是持久的。如果应用程序需要一个数据结构来对 XML 文档建模，则必须自己编写那样的代码。如果需要从 SAX 事件访问数据，并且没有把那个数据存储在代码中，那么用户不得不再次解析该文档。

3．JDOM

用 DOM 和 SAX 模型完成某些任务时的困难使 Jason Hunter 和 Brett McLaughlin 感到失望，于是他们创建了 JDOM 包。JDOM 是基于 Java 技术的开放源码项目，它试图遵循 80/20 规则：用 DOM 和 SAX 20%的功能来满足 80%的用户需求。JDOM 使用 SAX 和 DOM 解析器，因此，它是作为一组相对较小的 Java 类被实现的。

JDOM 的主要特性是它极大地减少了用户必须编写的代码数量，JDOM 应用程序的长度通常是 DOM 应用程序的 1/3，大约是 SAX 应用程序的一半。JDOM 并不做所有的事，但对于大多数用户要做的解析，它可能正好适合用户的需求。

4．JAXP

尽管 DOM、SAX 和 JDOM 为大多数常见任务提供了标准接口，但仍有些事情是它们不能解决的。例如，在 Java 程序中创建 DOMParser 对象的过程因 DOM 解析器的不同而不同。为了修正这个问题，SUN 发布了 JAXP，该 API 为使用 DOM、SAX 和 XSLT 处理 XML 文档提供了公共接口。

JAXP 提供的诸如 DocumentBuilderFactory 和 DocumentBuilder 之类的接口，为不同

的解析器提供了一个标准接口。还有一些方法可以允许用户控制底层的解析器是否可以识别名称空间，以及是否使用 DTD 或模式来验证 XML 文档。

5．接口的选择

在实际应用中，为了选择合适的接口类型，需要理解所有接口的设计要点，而且需要理解应用程序用 XML 档来做什么。考虑下面的问题将有助于找到正确的方法。

（1）要用 Java 编写应用程序吗？JAXP 使用 DOM、SAX 和 JDOM；如果用 Java 编写代码，那么应使用 JAXP 将代码与各种解析器实现的细节隔离。

（2）应用程序将如何部署？如果应用程序将要作为 Java applet 部署，那么会希望使要下载的代码数量最小，SAX 解析器比 DOM 解析器小，而使用 JDOM 时，除了 SAX 或 DOM 解析器之外还要求编写少量的代码。

（3）一旦解析了 XML 文档，还需要多次访问那些数据吗？如果需要回过头来访问 XML 文件的已解析版本，DOM 可能是正确的选择。而 SAX 事件被触发时，如果以后需要它，则由（开发人员）自己决定以某种方式保存它。如果需要访问不曾保存的事件，则必须再次解析该文件；而 DOM 自动保存所有的数据。

（4）只需要 XML 源文件的少量内容吗？如果只需要 XML 源文件的少量内容，那么 SAX 可能是正确的选择。SAX 不会为源文件中的每个东西创建对象。使用 SAX，要检查每个事件以了解它是否与需要有关，然后相应地处理它。

（5）正在一台内存很少的机器上工作吗？若是的话，不管可能考虑到的其他因素是什么，SAX 都是最佳选择。

10.3.2 XML 开发工具

支持 XML 的开发工具比较多，开发语言有脚本语言（如 JavaScript、VBScript、Perl 等）、编程语言（如 Java、C++、Object Pascal 等），开发环境有命令行工具（如 ANT 等）、集成式开发平台（如 VisualStudio.NET、VisualAge、Inprise JBuilder 等），建模工具有 IBM Rose 等。

下面是一个用脚本语言 JavaScript 处理 DOM 的例子，用于在 HTML 中简单地显示前面提到过的 visit 数据：

```
<?xml version="1.0"?>
<visit>
        <to>alluser</to>
        <from>educity.cn</from>
        <heading>Welcome</heading>
        <body>Welcome to the best online education website!</body>
</visit>
```

visit.htm 的内容：

```
<html>
<head>
<script language="JavaScript" for="window" event="onload">
        var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
        xmlDoc.async="false";
        xmlDoc.load("visit.xml");
        nodes = xmlDoc.documentElement.childNodes;
        to.innerText = nodes.item(0).text;
        from.innerText = nodes.item(1).text;
        header.innerText = nodes.item(2).text;
        body.innerText = nodes.item(3).text;
</script>
<title>HTML using XML data</title>
</head>
<body bgcolor="yellow">
        <h1>Refsnes Data Internal Note</h1>
        <b>To: </b>
        <span id="to"> </span>
        <br>
        <b>From: </b>
        <span id="from"></span>
        <hr>
        <b><span id="header"></span></b>
        <hr>
        <span id="body"></span>
</body>
</html>
```

10.3.3　XML 建模

与 XML 开发相关，而且可能比开发更重要的工作，是 XML 建模。XML 技术的出现，使用户对信息的描述能力增强了。它使得用户得以设计不会随着关键数据变化而要修改代码的程序，提高系统的可维护性和适应性，构造与平台无关的数据，构造可以在不同系统中使用和交换的数据，定义在不同的部门共同遵守的信息标准。但要享受到 XML 带来的惊喜，就必须很好地掌握怎样用 XML 来描述信息，也就是 XML 建模。

XML 并不是程序设计层面的技术。用户使用 XML 建模的能力越强，它给用户带来的回报就越多。XML 建模包括以下方面：

（1）描述具体数据：这是 XML 的经典用途，可以用任何文本编辑工具来编写 XML 文档。

（2）描述数据结构和模式：这需要用到 DTD。

（3）描述数据的表现：XSL 定义了一组元素（称为格式化对象），它们描述应该如何格式化数据；XSLT 是一个描述如何将 XML 文档转换成别的东西的 XML 词汇表。

（4）描述数据中的位置：XML 路径语言（XML Path Language，XPath）是描述 XML 文档中位置的语法。使用 XSLT 样式表中的 XPath 来描述用户希望转换 XML 档的哪个部分。XPath 也用在其他 XML 标准中，这就是为什么它是独立于 XSLT 的标准的原因。

（5）描述数据中的链接关系：XML 链接语言（XML Linking Language，XLink）定义将不同资源链接在一起的各种方法。用户可以进行正常的点对点链接（就像用 HTML <a> 元素）或扩展的链接，后者可包括多点链接、通过第三方的链接以及定义转向给定链接的意义的规则。XPointer 使用 XPath 作为引用其他资源的方法，它还包括对 XPath 的一些扩展。

（6）描述数据的应用关系：通过 SOAP、WSDL、UDDI 来描述。

XML 技术为信息规划和信息系统分析人员扩展了视野，但独立的、简单易用的 XML 建模工具并不多见，这些工具大多嵌入在其他强大的建模工具或是开发平台中。

本章参考文献

[1] 孙一中. XML 理论和应用基础. 北京：北京邮电大学出版社，2001
[2] 杜大鹏，李善茂，李珊蓉等. XML 实用大全. 北京：中国水利水电出版社，2000
[3] 孔梦荣，韩玉民. XML 基础教程. 北京：清华大学出版社，2008

第 11 章　软 件 架 构

20 世纪 60 年代的软件危机使得人们开始重视软件工程的研究。起初，人们把软件设计的重点放在数据结构和算法的选择上，随着软件系统规模越来越大、越来越复杂，整个系统的结构和规格说明显得越来越重要。随着软件危机的程度日益加剧，现有的软件工程方法对此显得力不从心。对于大规模的复杂软件系统来说，对总体的系统结构设计和规格说明比起对计算的算法和数据结构的选择已经变得明显重要得多。在此种背景下，人们认识到软件架构（Software Architecture，SA）的重要性，并认为对软件架构的系统、深入的研究将会成为提高软件生产率和解决软件维护问题的新的最有希望的途径。

11.1　软件架构概述

软件架构虽脱胎于软件工程，但其形成同时借鉴了计算机架构和网络架构中很多宝贵的思想和方法，最近几年软件架构研究已完全独立于软件工程的研究，成为计算机科学的一个最新的研究方向和独立学科分支。软件架构研究的主要内容涉及软件架构描述、软件架构风格、软件架构评价和软件架构的形式化方法等。解决好软件的重用、质量和维护问题，是研究软件架构的根本目的。

1．软件架构的定义

虽然软件架构已经在软件工程领域中有着广泛的应用，但迄今为止还没有一个被大家所公认的定义。许多专家学者从不同角度和不同侧面对软件架构进行了刻画，在本书中，如果不特别指出，将使用软件架构的下列定义：

软件架构为软件系统提供了一个结构、行为和属性的高级抽象，由构成系统的元素的描述、这些元素的相互作用、指导元素集成的模式以及这些模式的约束组成。软件架构不仅指定了系统的组织（organization）结构和拓扑（topology）结构，并且显示了系统需求和构成系统的元素之间的对应关系，提供了一些设计决策的基本原理。

2．软件架构的意义

对于软件项目的开发来说，一个清晰的软件架构是首要的。传统的软件开发过程可以划分为从概念直到实现的若干个阶段，包括问题定义、需求分析、软件设计、软件实现及软件测试等。软件架构的建立应位于需求分析之后，软件设计之前。但在传统的软件工程方法中，需求和设计之间存在一条很难逾越的鸿沟，从而很难有效地将需求转换为相应的设计。而软件架构就是试图在软件需求与软件设计之间架起一座桥梁，着重解决软件系统的结构和需求向实现平坦地过渡的问题。

软件架构是风险承担者进行交流的手段，明确了对系统实现的约束条件，决定了开发和维护组织的组织结构，制约着系统的质量属性。软件架构使推理和控制更改更简单，有助于循序渐进的原型设计，可以作为培训的基础。软件架构是可传递和可重用的模型，通过研究软件架构可能预测软件的质量。

3．软件架构的发展史

软件系统的规模在迅速增大的同时，软件开发方法也经历了一系列的变革。在此过程中，软件架构也由最初模糊的概念发展到一个渐趋成熟的理论和技术。

20 世纪 70 年代以前，尤其是在以 ALGOL 68 为代表的高级语言出现以前，软件开发基本上都是汇编程序设计，此阶段系统规模较小，很少明确考虑系统结构，一般不存在系统建模工作。70 年代中后期，由于结构化开发方法的出现与广泛应用，软件开发中出现了概要设计与详细设计，而且主要任务是数据流设计与控制流设计，因此，此时软件结构已作为一个明确的概念出现在系统的开发中。

20 世纪 80 年代初到 90 年代中期，是面向对象开发方法兴起与成熟阶段。由于对象是数据与基于数据之上操作的封装，因而在面向对象开发方法下，数据流设计与控制流设计则统一为对象建模，同时，面向对象方法还提出了一些其他的结构视图。如在对象建模技术（Object Modeling Technology，OMT）方法中提出了功能视图、对象视图与动态视图（包括状态图和事件追踪图）；而 Booch 方法中则提出了类视图、对象视图、状态迁移图、交互作用图、模块图、进程图；而 1997 年出现的统一建模语言（Unified Modeling Language，UML）则从功能模型、静态模型、动态模型、配置模型等方面描述应用系统的结构。

20 世纪 90 年代以后，则是基于构件的软件开发阶段，该阶段以过程为中心，强调软件开发采用构件化技术和架构技术，要求开发出的软件具备很强的自适应性、互操作性、可扩展性和可重用性。此阶段中，软件架构已经作为一个明确的文档和中间产品存在于软件开发过程中，同时，软件架构作为一门学科逐渐得到人们的重视，并成为软件工程领域的研究热点。

纵观软件架构技术的发展过程，从最初的“无架构”设计到现行的基于架构的软件开发，可以认为经历了 4 个阶段：

（1）“无架构”设计阶段。以汇编语言进行小规模应用程序开发为特征。

（2）萌芽阶段。出现了程序结构设计主题，以控制流图和数据流图构成软件结构为特征。

（3）初期阶段。出现了从不同侧面描述系统的结构模型，以 UML 为典型代表。

（4）高级阶段。以描述系统的高层抽象结构为中心，不关心具体的建模细节，划分了架构模型与传统软件结构的界限，该阶段以 Kruchten 提出的“4+1”模型为标志。

11.2　软件架构建模

设计软件架构的首要问题是如何表示软件架构，即如何对软件架构建模。根据建模的侧重点不同，可以将软件架构的模型分为 5 种：结构模型、框架模型、动态模型、过程模型和功能模型。在这 5 个模型中，最常用的是结构模型和动态模型。

（1）结构模型：这是一个最直观、最普遍的建模方法。这种方法以架构的构件、连接件（connector）和其他概念来刻画结构，并力图通过结构来反映系统的重要语义内容，包括系统的配置、约束、隐含的假设条件、风格、性质等。研究结构模型的核心是架构描述语言。

（2）框架模型：框架模型与结构模型类似，但它不太侧重描述结构的细节而更侧重于整体的结构。框架模型主要以一些特殊的问题为目标建立只针对和适应该问题的结构。

（3）动态模型：动态模型是对结构或框架模型的补充，研究系统的“大颗粒”的行为性质。例如，描述系统的重新配置或演化。动态可以指系统总体结构的配置、建立或拆除通信通道或计算的过程。这类系统通常是激励型的。

（4）过程模型：过程模型研究构造系统的步骤和过程。因而结构是遵循某些过程脚本的结果。

（5）功能模型：该模型认为架构是由一组功能构件按层次组成，下层向上层提供服务。它可以看作是一种特殊的框架模型。

上述 5 种模型各有所长，将 5 种模型有机地统一在一起，形成一个完整的模型来刻画软件架构更合适。例如，Kruchten 在 1995 年提出了一个“4+1”的视图模型，如图 11-1 所示。

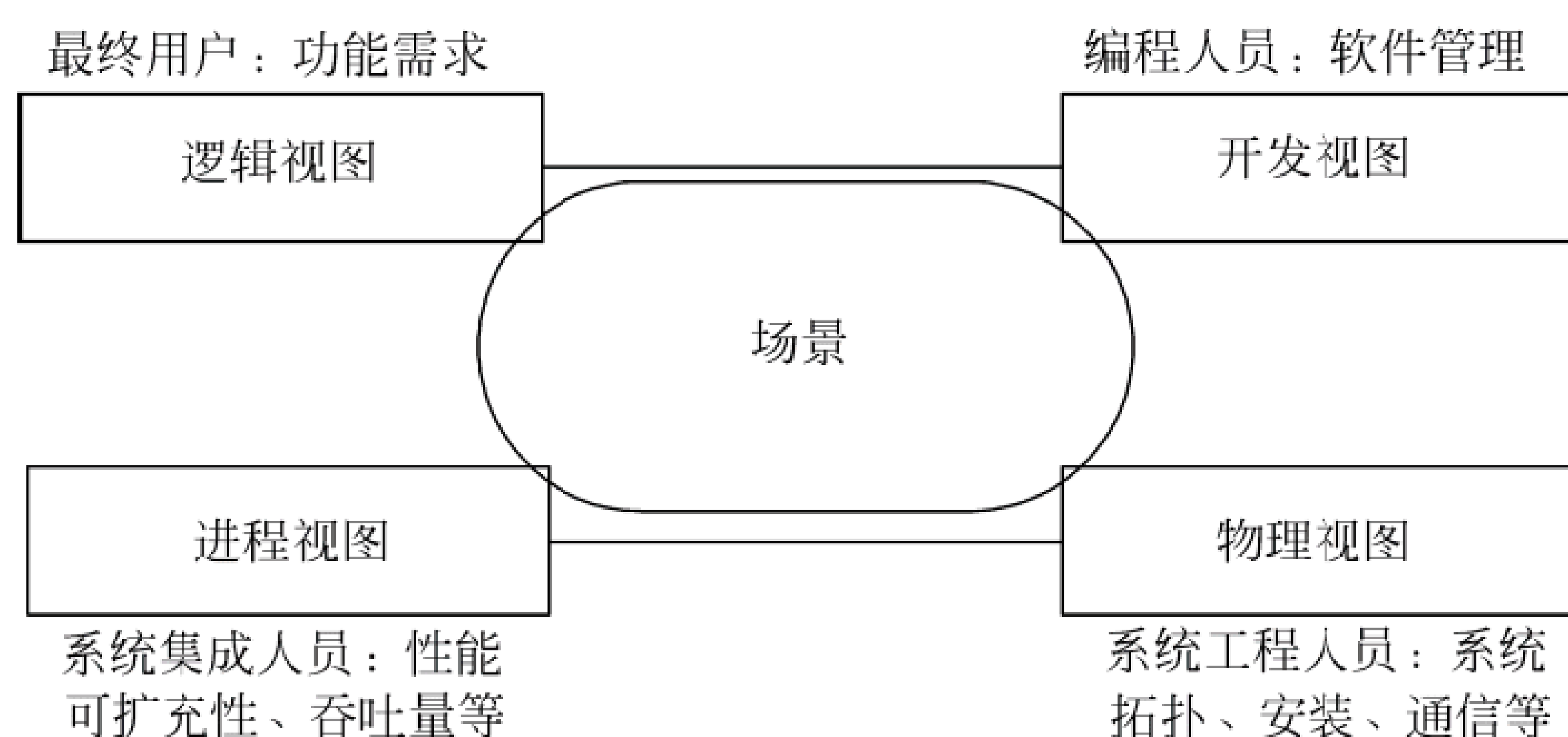


图 11-1　“4+1”视图模型

“4+1”视图模型从 5 个不同的视角包括逻辑视图、进程视图、物理视图、开发视图和场景视图来描述软件架构。每一个视图只关心系统的一个侧面，5 个视图结合在一起才能反映系统的软件架构的全部内容。

11.2.1 逻辑视图

逻辑视图（Logic View）主要支持系统的功能需求，即系统提供给最终用户的服务。在逻辑视图中，系统分解成一系列的功能抽象，这些抽象主要来自问题领域。这种分解不但可以用来进行功能分析，而且可用作标识在整个系统的各个不同部分的通用机制和设计元素。在面向对象技术中，通过抽象、封装和继承，可以用对象模型来代表逻辑视图，用类图（Class Diagram）来描述逻辑视图。可以从 Booch 标记法中导出逻辑视图的标记法，只是从架构级的范畴来考虑这些符号，用 Rational Rose 进行架构设计。图 11-2 是逻辑视图中使用的符号集合。

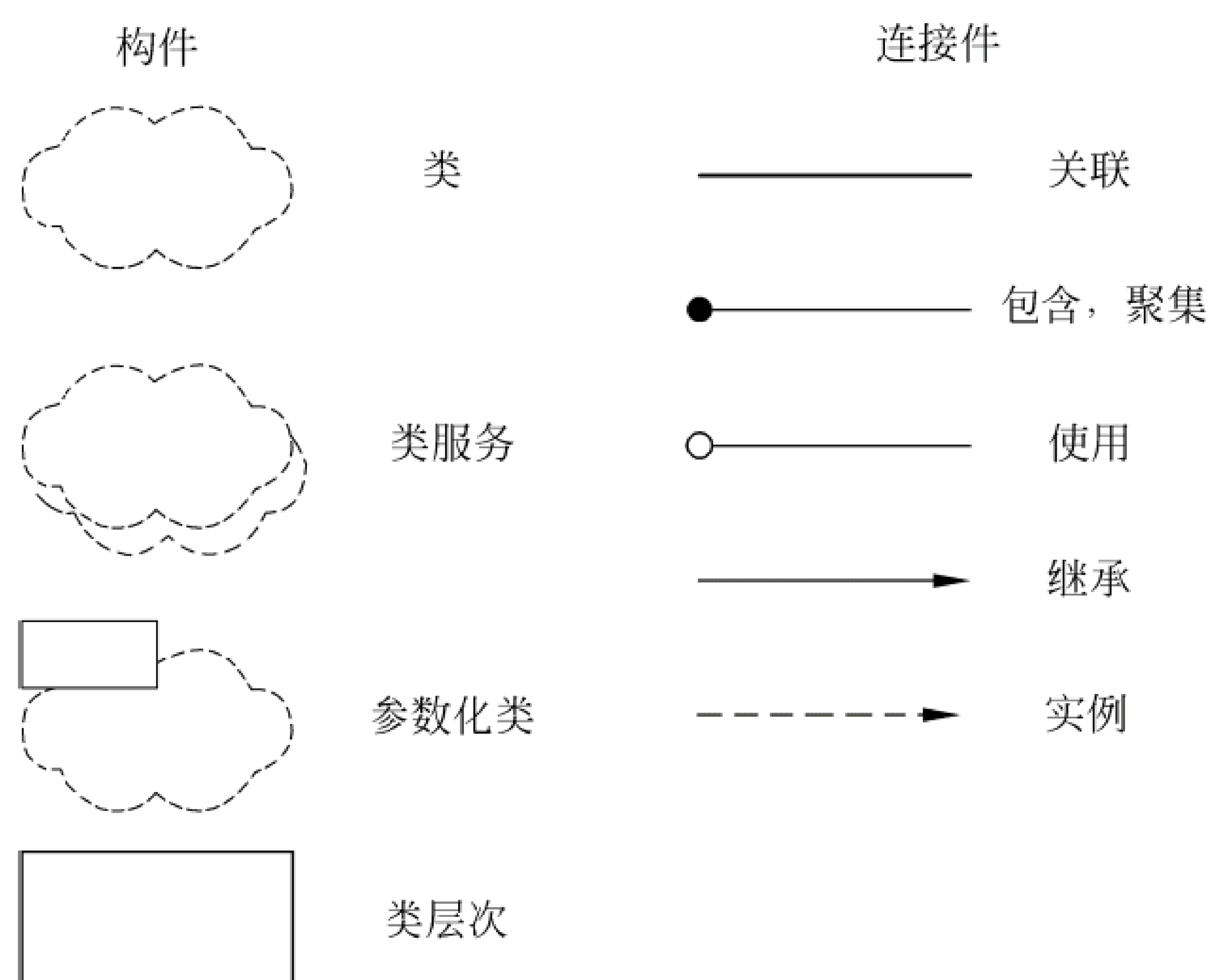


图 11-2 逻辑视图中使用的标记符号

类图用于表示类的存在以及类与类之间的相互关系，是从系统构成的角度来描述正在开发的系统。一个类的存在不是孤立的，类与类之间以不同方式互相合作，共同完成某些系统功能。关联关系表示两个类之间存在着某种语义上的联系，其真正含义要由附加在横线之上的一个短语来予以说明。在表示包含关系的图符中，带有实心圆的一端表示整体，相反的一端表示部分。在表示使用关系的图符中，带有空心圆的一端连接在请求服务的类，相反的一端连接在提供服务的类。在表示继承关系的图符中，箭头由子类指向基类。

逻辑视图中使用的风格为面向对象的风格，逻辑视图设计中要注意的主要问题是要保持一个单一的、内聚的对象模型贯穿整个系统。例如，图 11-3 是某通信系统架构（ACS）中的主要类。

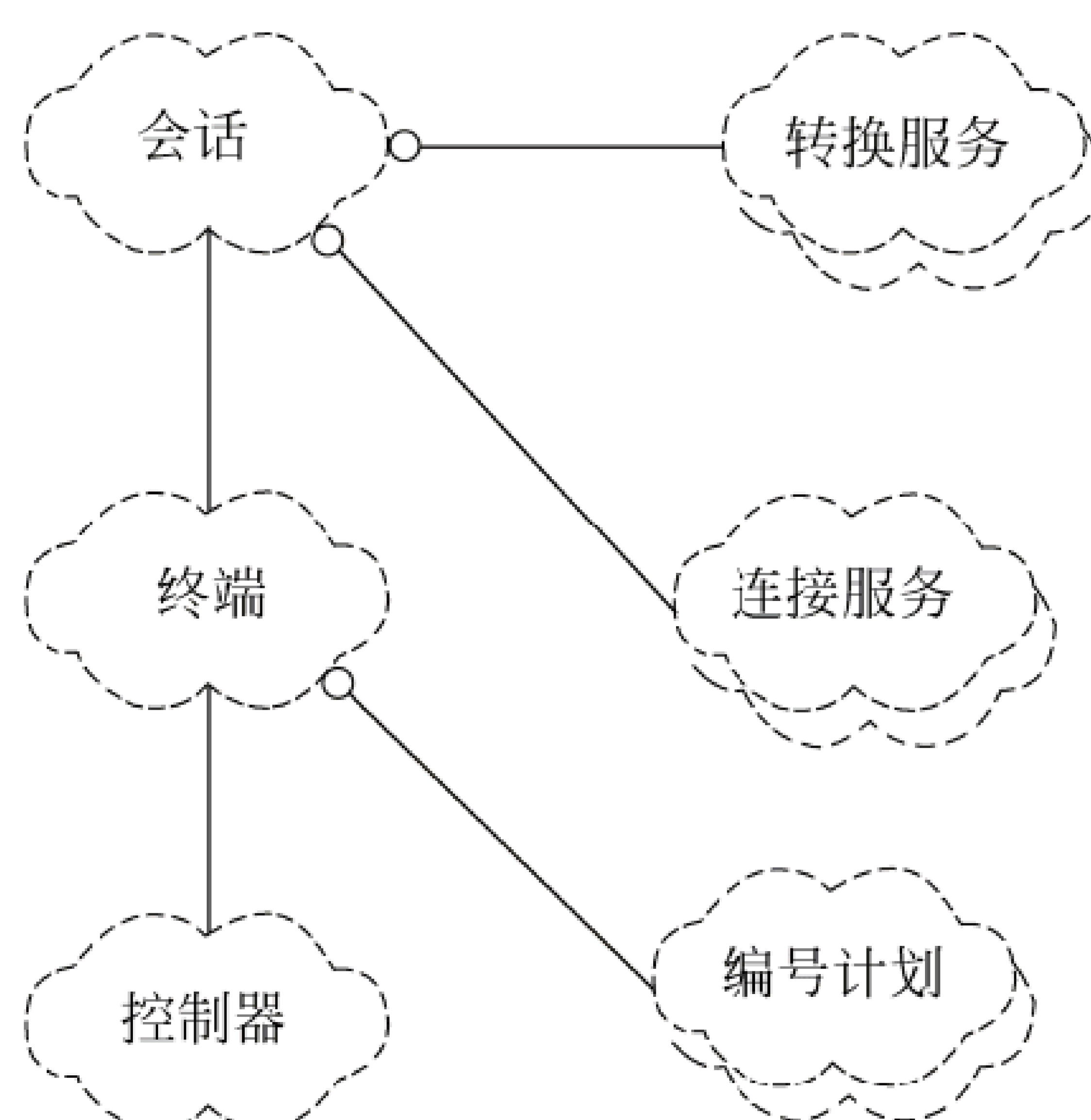


图 11-3　某通信系统架构逻辑视图

ACS 的功能是在终端之间建立连接，这种终端可以是电话机、主干线、专用线路、特殊电话线、数据线等，不同线路由不同的线路接口卡进行支持。线路控制器对象的作用是译码并把所有符号加入到线路接口卡中。终端对象的作用是保持终端的状态，代表本条线路的利益参与协商服务。会话对象代表一组参与会话的终端，使用转换服务（目录、逻辑地址映射到物理地址，路由等）和连接服务在终端之间建立语音路径。

11.2.2　开发视图

开发视图（Development View）也称模块视图（Module View），主要侧重于软件模块的组织和管理。软件可通过程序库或子系统进行组织，这样，对于一个软件系统，就可以由不同的人进行开发。开发视图要考虑软件内部的需求，如软件开发的容易性、软件的重用性和软件的通用性，要充分考虑由于具体开发工具的不同而带来的局限性。

开发视图通过系统输入输出关系的模型图和子系统图来描述。可以在确定了软件包含的所有元素之后描述完整的开发角度，也可以在确定每个元素之前，列出开发视图原则。

与逻辑视图一样，可以使用 Booch 标记法中某些符号来表示开发视图，如图 11-4 所示。

在开发视图中，最好采用 4～6 层子系统，而且每个子系统仅仅能与同层或更低层的子系统通信，这样可以使每个层次的接口既完备又精练，避免了各个模块之间很复杂的依赖关系。而且，设计时要充分考虑，对于各个层次，层次越低，通用性越强，这样，可以保证应用程序的需求发生改变时，所做的改动最小。开发视图所用的风格通常是层次结构风格。例如，图 11-5 表示的是空中交通管制系统的五层结构图。

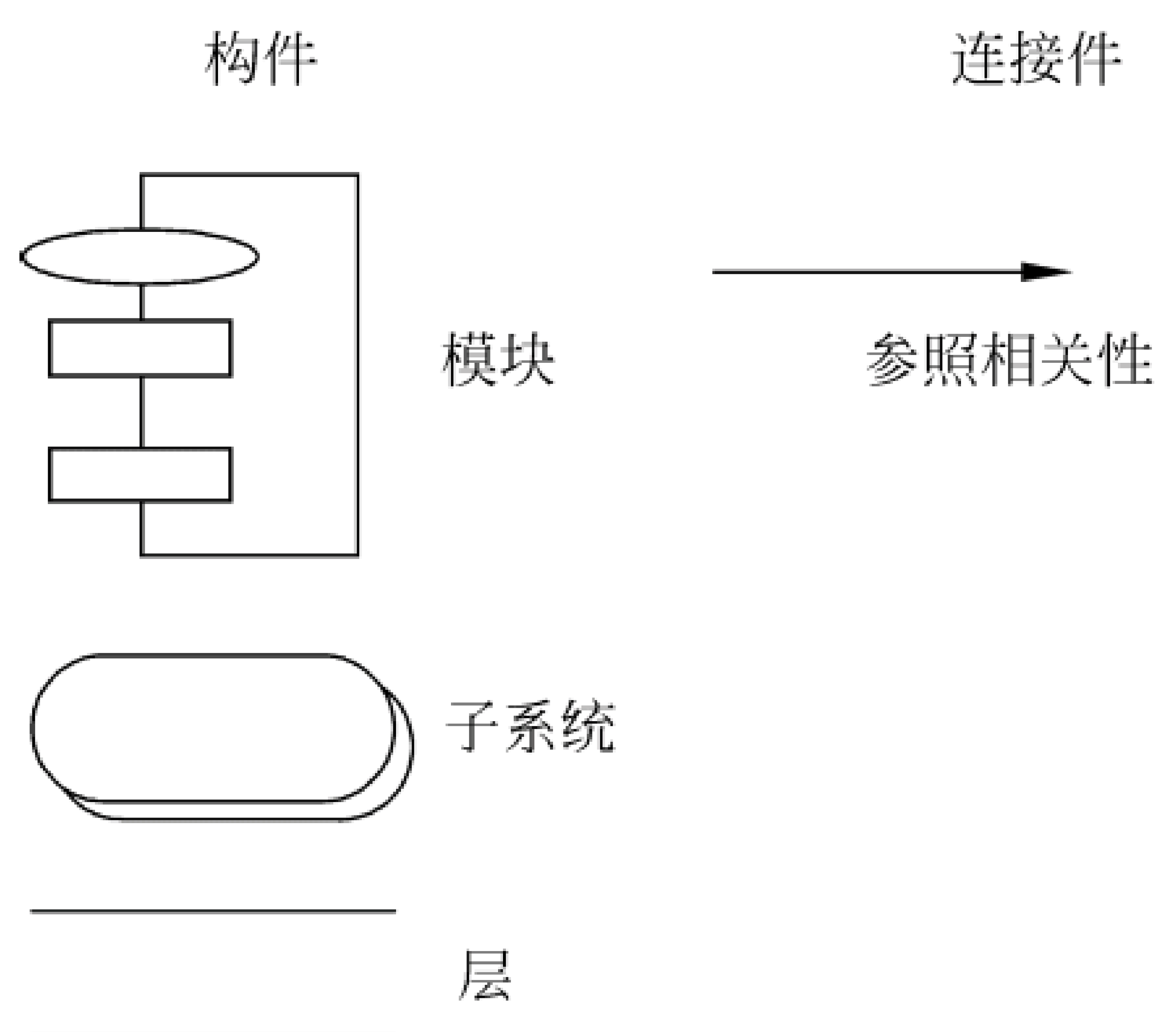


图 11-4 开发视图中使用的标记符号

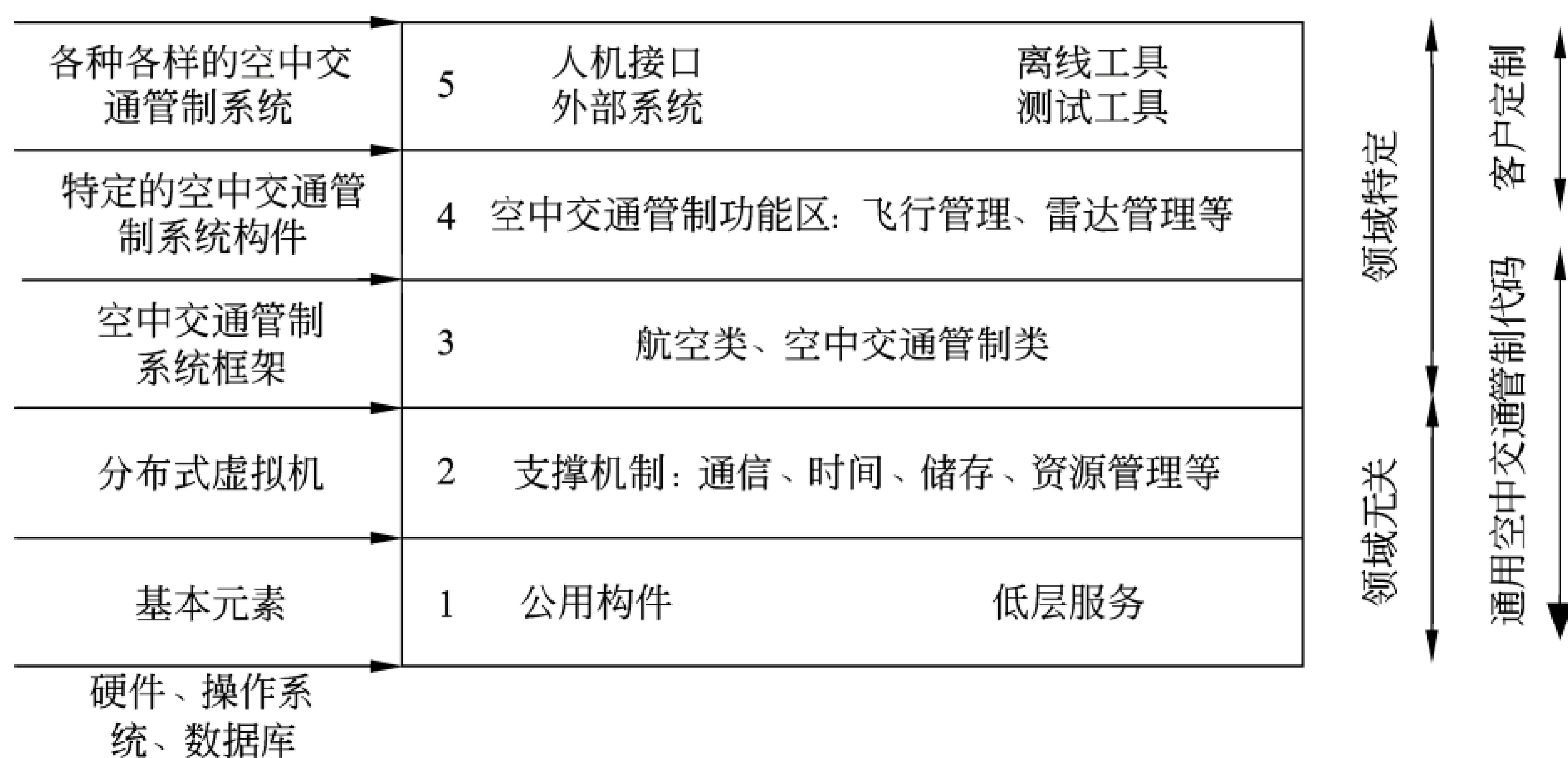


图 11-5 空中交通管制系统的五层结构

图 11-5 是图 11-3 的开发视图。第 1 层和第 2 层组成了一个领域无关的分布式基础设施，贯穿于整个产品线中，并且与硬件平台、操作系统或数据库管理系统等无关。第 3 层增加了空中交通管制系统的框架，以形成一个领域特定的软件架构。第 4 层使用该框架建立一个功能平台，第 5 层则依赖于具体客户和产品，包含了大部分用户接口以及与外部系统的接口。

11.2.3 进程视图

进程视图（Process View）侧重于系统的运行特性，主要关注一些非功能性的需求，例如，系统的性能和可用性。进程视图强调并发性、分布性、系统集成性和容错能力，以及从逻辑视图中的主要抽象如何适合进程结构。它也定义逻辑视图中的各个类的操作具体是在哪一个线程（thread）中被执行的。

进程视图可以描述成多层抽象，每个级别分别关注不同的方面。在最高层抽象中，进程结构可以看作是构成一个执行单元的一组任务。它可看成一系列独立的，通过逻辑

网络相互通信的程序。它们是分布的，通过总线或局域网、广域网等硬件资源连接起来。通过进程视图可以从进程测量一个目标系统最终执行情况。例如在以计算机网络作为运行环境的图书管理系统中，服务器需对来自各个不同的客户机的进程管理，决定某个特定进程（如查询子进程、借还书子进程）的唤醒、启动、关闭等操作，从而控制整个网络协调有序地工作。

通过扩展 Booch 对 Ada 任务的表示法，来表示进程视图，从架构角度来看，进程视图的标记元素如图 11-6 所示。

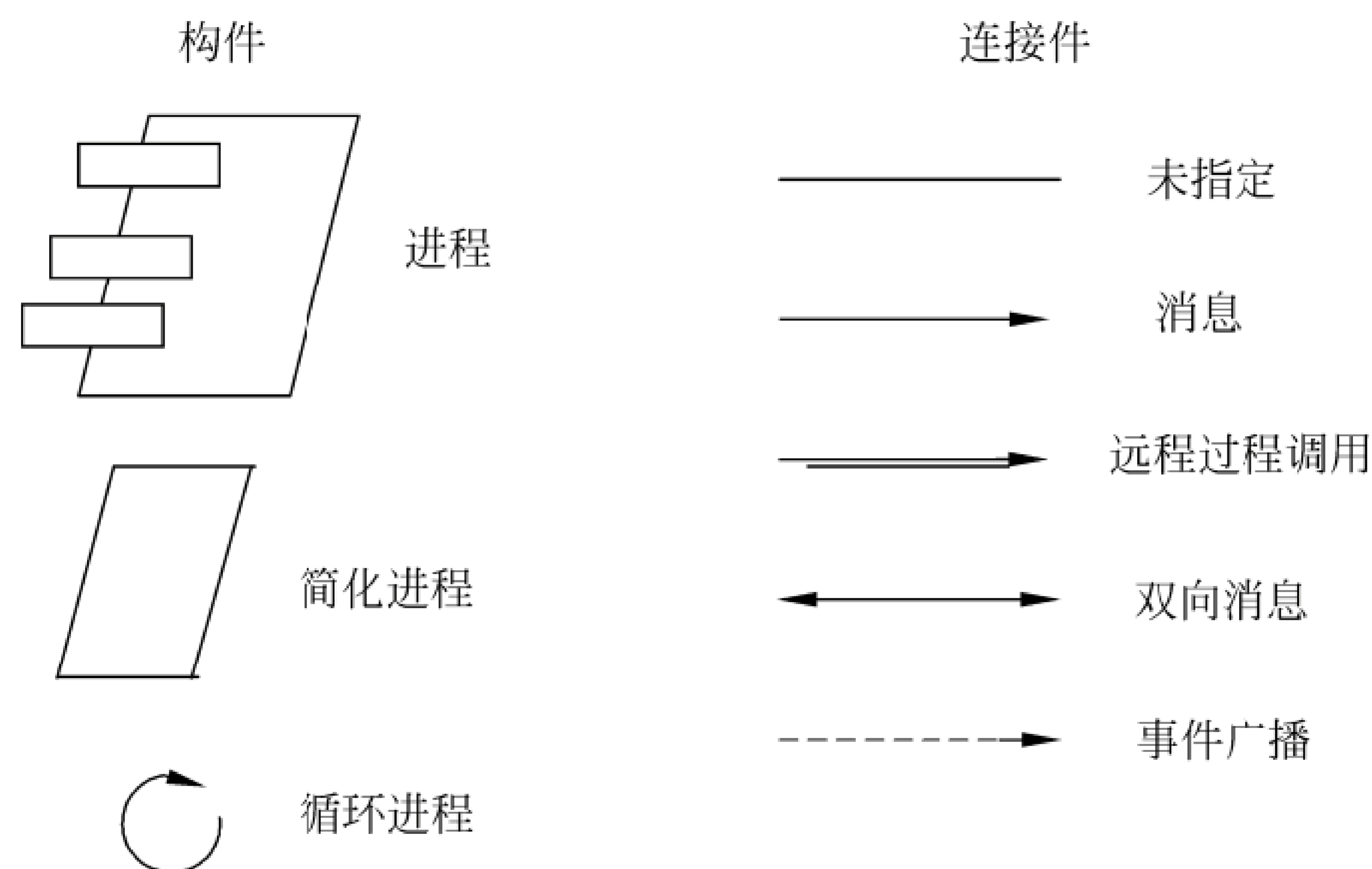


图 11-6　进程视图中使用的标记符号

有很多风格适用于进程视图，例如，管道和过滤器风格、客户/服务器风格（多客户/单服务器，多客户/多服务器）等。图 11-7 是 ACS 系统的进程视图（局部）。

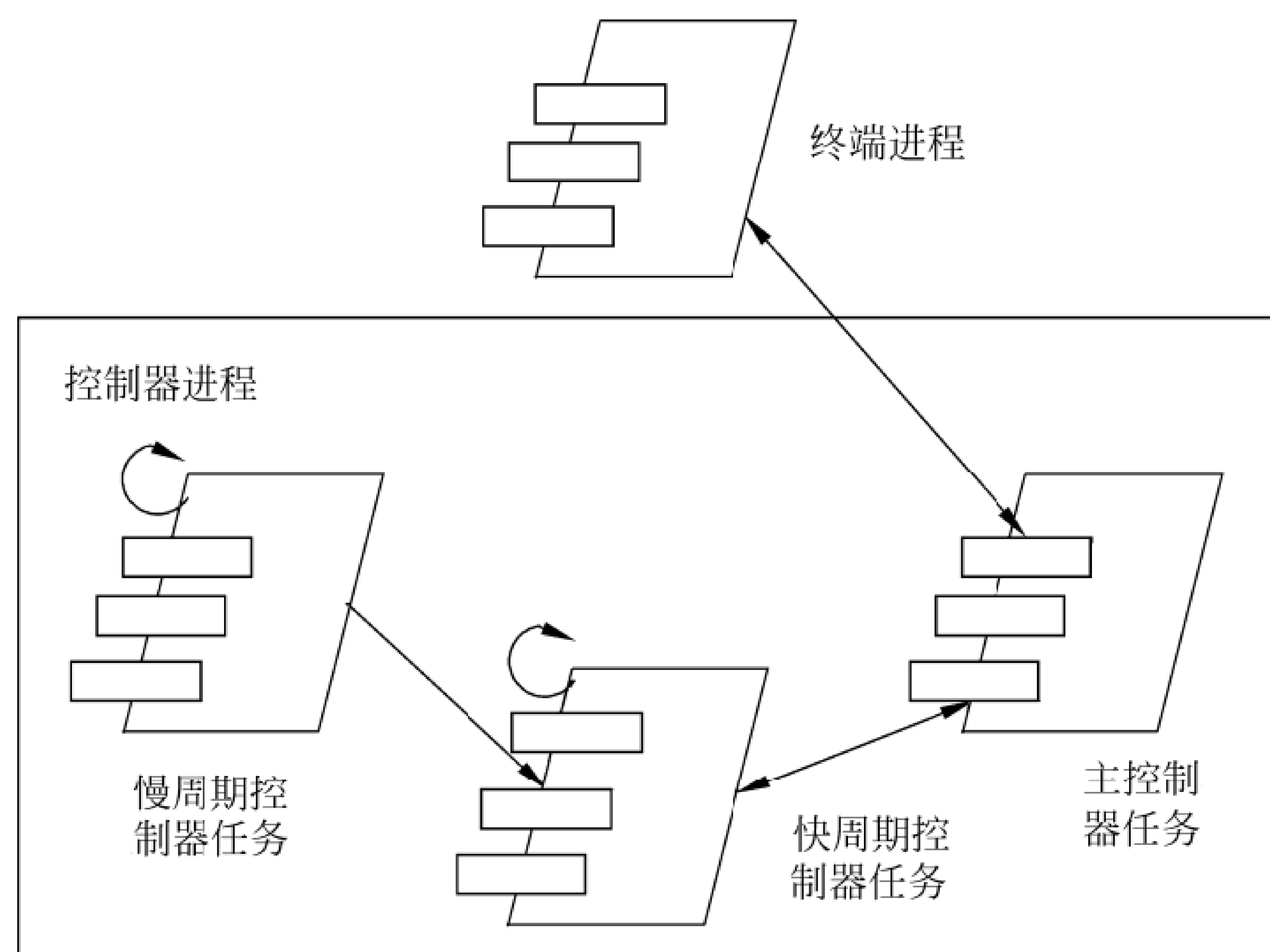


图 11-7　ACS 系统的进程视图（局部）

在图 11-7 中，所有终端均由同一个终端进程进行处理，由其输入队列中的消息驱动。控制器对象在组成控制器进程的三个任务之一中执行，慢循环周期（200ms）任务扫描所有挂起（suspend）终端，把任何一个活动的终端置入快循环周期（10ms）任务的扫描列表，快循环周期任务检测任何显著的状态改变，并把改变的状态传递给主控制器任务，主控制器任务解释改变，通过消息与相应的终端进行通信。在这里，通过共享内存实现在控制器进程中传递的消息。

11.2.4 物理视图

物理视图（Physical View，PV）主要考虑如何把软件映射到硬件上，它通常要考虑到系统性能、规模、可靠性等。解决系统拓扑结构、系统安装、通信等问题。当软件运行于不同的节点上时，各视图中的构件都直接或间接地对应于系统的不同节点上。因此，从软件到节点的映射要有较高的灵活性，当环境改变时，对系统其他视图的影响最小。

大型系统的物理视图可能会变得十分混乱，因此可以与进程视图的映射一道，以多种形式出现，也可单独出现。图 11-8 是物理视图的标记元素集合。

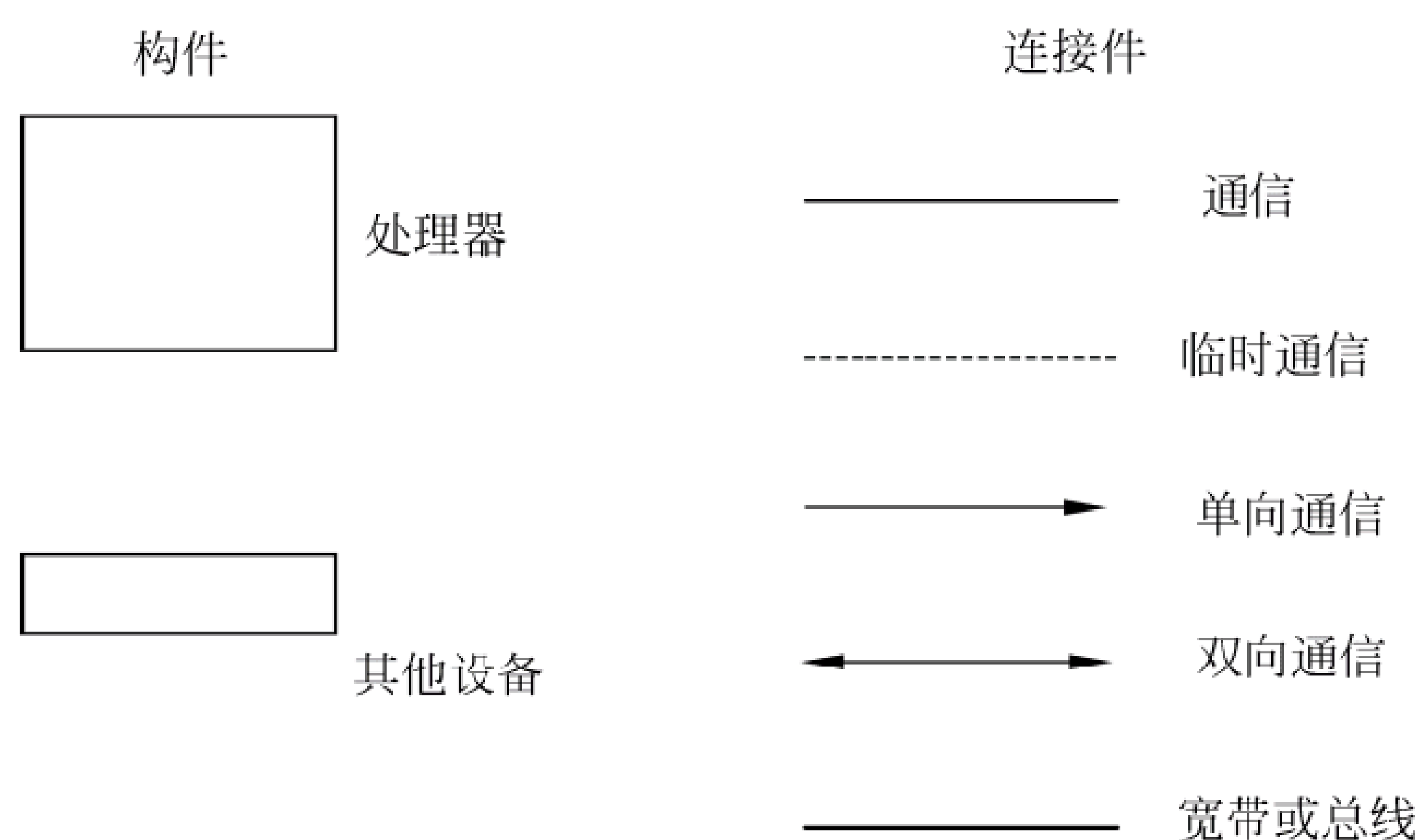


图 11-8 物理视图中使用的标记符号

图 11-9 是一个大型 ACS 系统的可能硬件配置，图 11-10 和图 11-11 是进程视图的两个不同的物理视图映射，分别对应一个小型的 ACS 和大型的 ACS，C、F 和 K 是三个不同容量的计算机类型，支持三个不同的可执行文件。

11.2.5 场景

场景（scenarios）可以看作是那些重要系统活动的抽象，它使四个视图有机联系起来，从某种意义上说场景是最重要的需求抽象。在开发架构时，它可以帮助设计者找到架构的构件和它们之间的作用关系。同时，也可以用场景来分析一个特定的视图，或描述不同视图构件间是如何相互作用的。场景可以用文本表示，也可以用图形表示。例如，图 11-12 是一个小型 ACS 系统的场景片段，相应的文本表示如下：

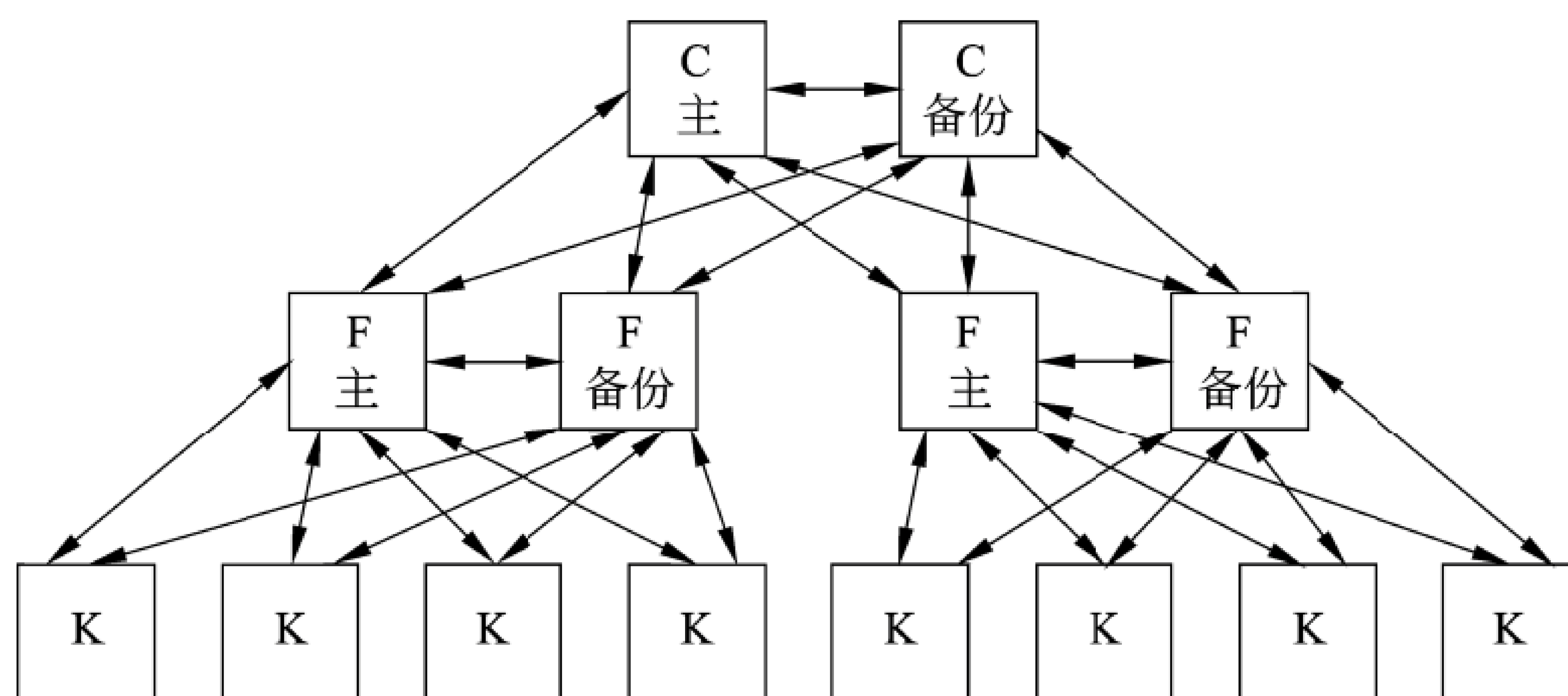


图 11-9　ACS 系统的物理视图

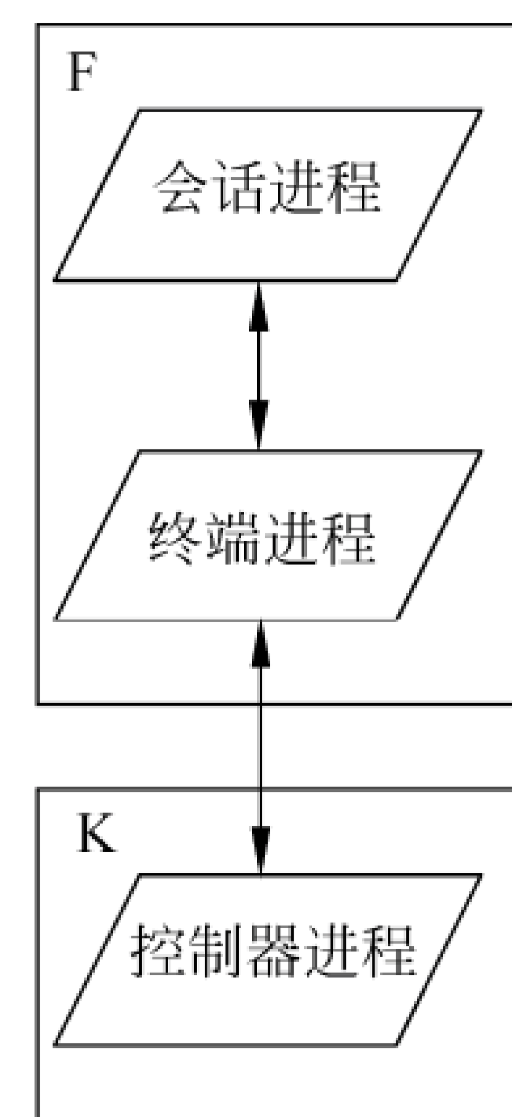


图 11-10　具有进程分配的小型 ACS 系统的物理视图

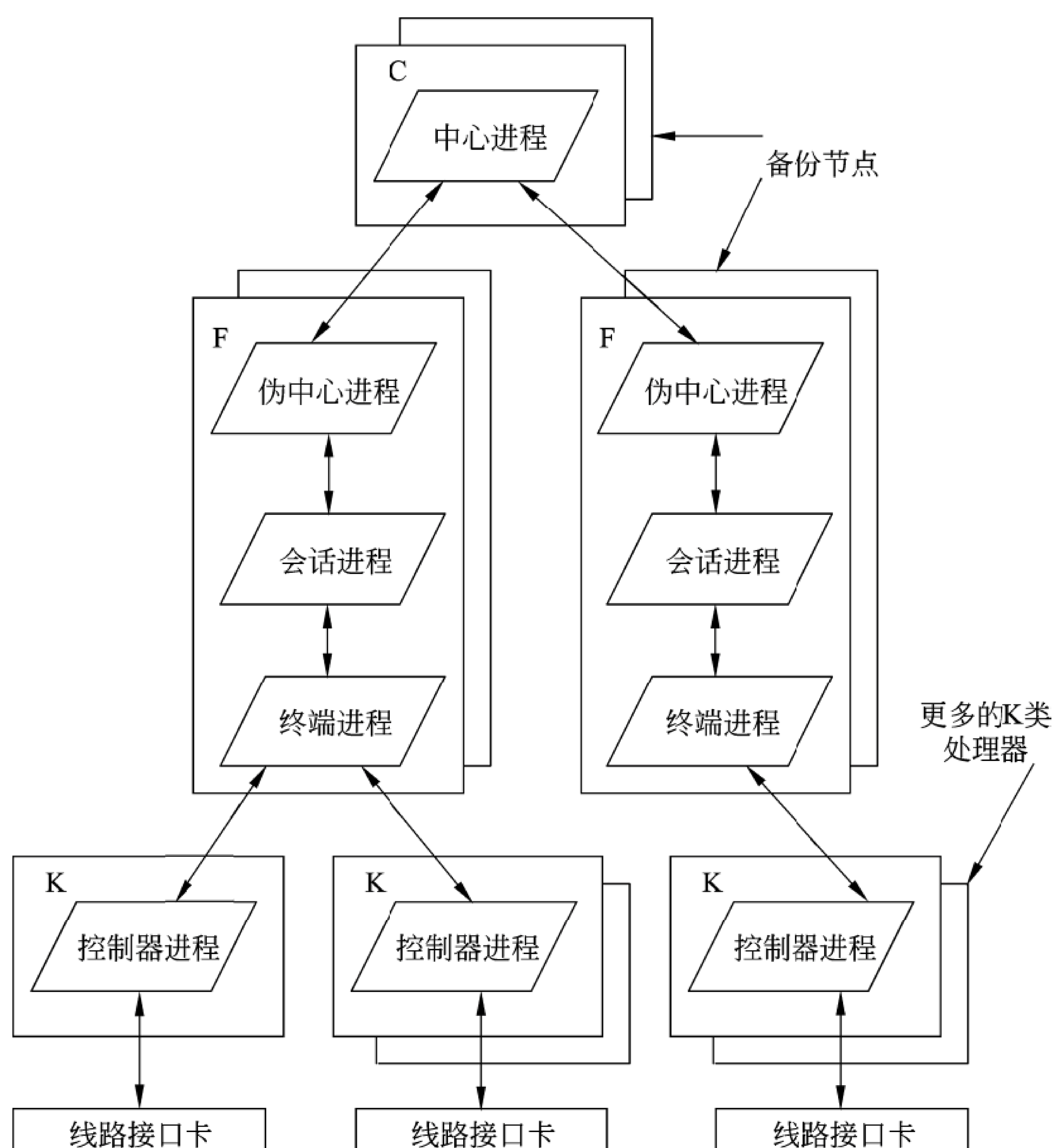


图 11-11　具有进程分配的大型 ACS 系统的物理视图

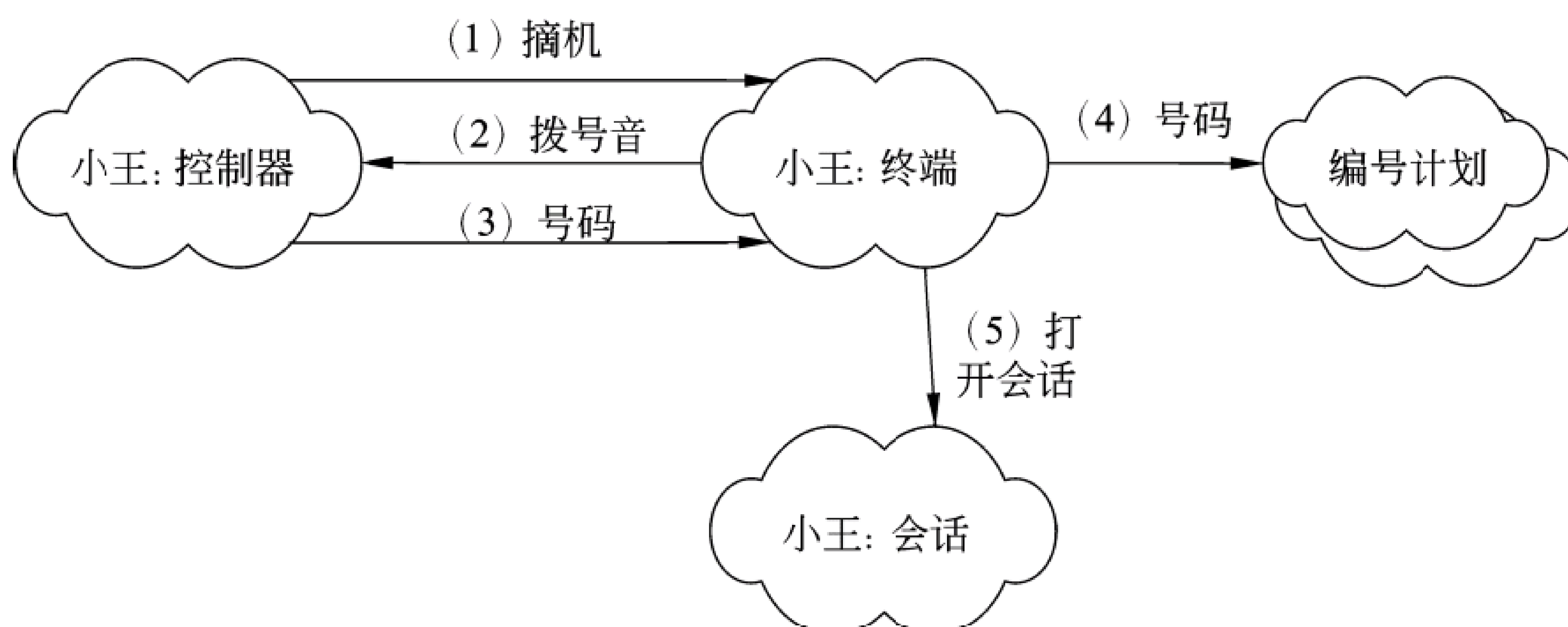


图 11-12　本地呼叫场景的一个原型

（1）小王的电话控制器检测和验证电话从挂机到摘机状态的转变，且发送一个消息以唤醒相应的终端对象。

（2）终端分配一定的资源，且通知控制器发出某种拨号音。

（3）控制器接收所拨号码并传给终端。

（4）终端使用编号计划分析号码。

（5）当一个有效的拨号序列进入时，终端打开一个会话。

从以上分析可知，逻辑视图和开发视图描述系统的静态结构，而进程视图和物理视图描述系统的动态结构。对于不同的软件系统来说，侧重的角度也有所不同。例如，对于管理信息系统来说，比较侧重于从逻辑视图和开发视图来描述系统，而对于实时控制系统来说，则比较注重于从进程视图和物理视图来描述系统。

11.3　软件架构风格

软件架构设计的一个核心问题是能否使用重复的架构模式，即能否达到架构级的软件重用。也就是说，能否在不同的软件系统中，使用同一架构。基于这个目的，学者们开始研究和实践软件架构的风格和类型问题。

软件架构风格是描述某一特定应用领域中系统组织方式的惯用模式（idiomatic paradigm）。架构风格定义了一个系统家族，即一个架构定义一个词汇表和一组约束。词汇表中包含一些构件和连接件类型，而这组约束指出系统是如何将这些构件和连接件组合起来的。架构风格反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个模块和子系统有效地组织成一个完整的系统。按这种方式理解，软件架构风格定义了用于描述系统的术语表和一组指导构件系统的规则。

对软件架构风格的研究和实践促进了对设计的重用，一些经过实践证实的解决方案也可以可靠地用于解决新的问题。架构风格的不变部分使不同的系统可以共享同一个实

现代码。只要系统是使用常用的、规范的方法来组织，就可使别的设计者很容易地理解系统的架构。例如，如果某人把系统描述为“客户/服务器”模式，则不必给出设计细节，立刻就会明白系统是如何组织和工作的。

Garlan 和 Shaw 根据此框架给出了通用架构风格的分类。

（1）数据流风格：批处理序列、管道/过滤器。

（2）调用/返回风格：主程序/子程序、面向对象风格、层次结构。

（3）独立构件风格：进程通信、事件系统。

（4）虚拟机风格：解释器、基于规则的系统。

（5）仓库风格：数据库系统、超文本系统、黑板系统。

11.3.1　分层系统

层次系统组织成一个层次结构，每一层为上层服务，并作为下层客户。在一些层次系统中，除了一些精心挑选的输出函数外，内部的层只对相邻的层可见。这样的系统中构件在一些层实现了虚拟机（在另一些层次系统中层是部分不透明的）。连接件通过决定层间如何交互的协议来定义，拓扑约束包括对相邻层间交互的约束。

这种风格支持基于可增加抽象层的设计。这样，允许将一个复杂问题分解成一个增量步骤序列的实现。由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样为软件重用提供了强大的支持。

图 11-13 是层次系统风格的示意图。层次系统最广泛的应用是分层通信协议。在这一应用领域中，每一层提供一个抽象的功能，作为上层通信的基础。较低的层次定义低层的交互，最低层通常只定义硬件物理连接。

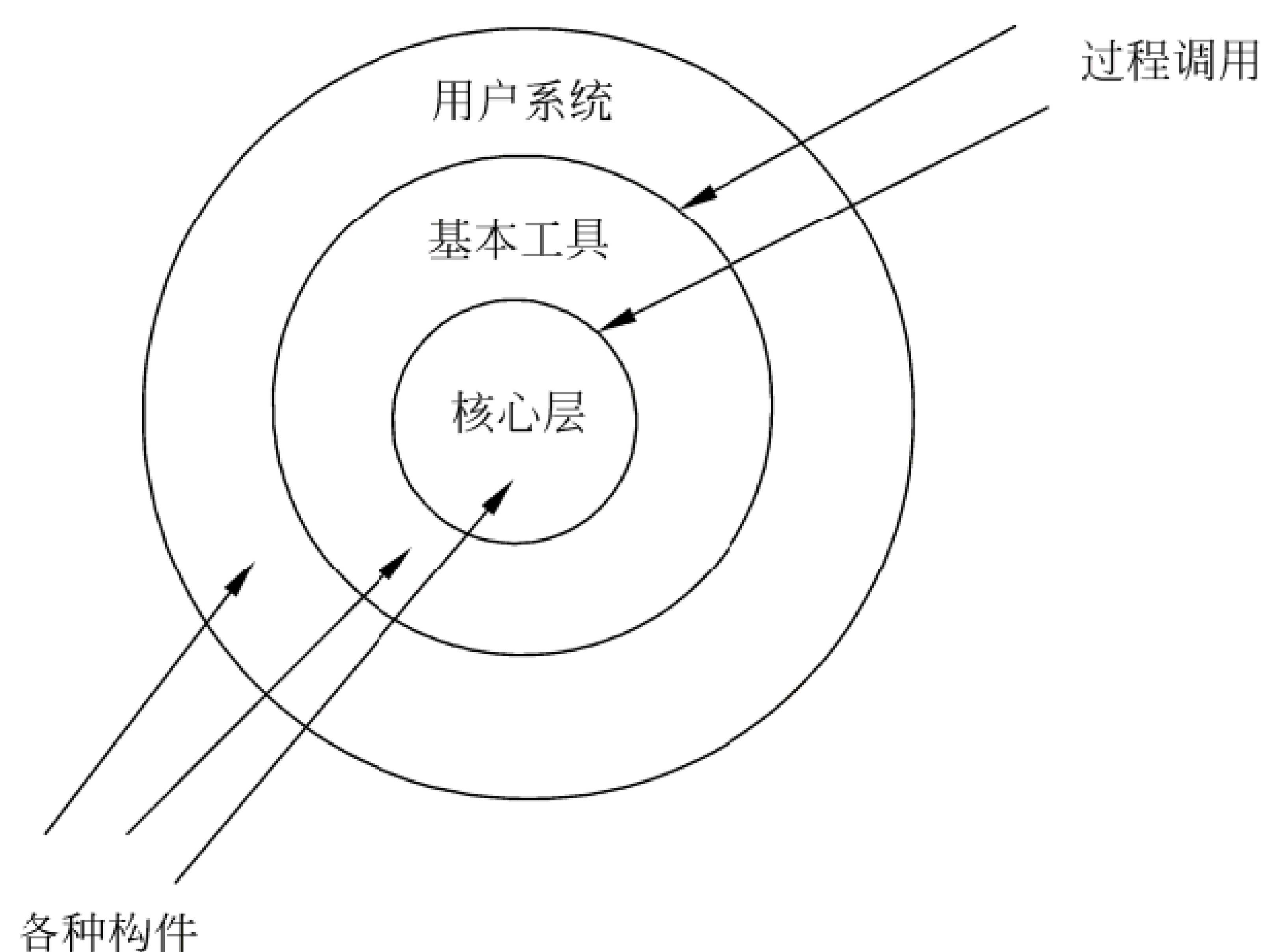


图 11-13　层次系统风格的架构

层次系统有许多可取的属性：

（1）支持基于抽象程度递增的系统设计，使设计者可以把一个复杂系统按递增的步骤进行分解。

（2）支持功能增强，因为每一层至多和相邻的上下层交互，因此功能的改变最多影响相邻的上下层。

（3）支持重用。只要提供的服务接口定义不变，同一层的不同实现可以交换使用。这样，就可以定义一组标准的接口，而允许各种不同的实现方法。

但是，层次系统也有其不足之处：

（1）并不是每个系统都可以很容易地划分为分层的模式，甚至即使一个系统的逻辑结构是层次化的，出于对系统性能的考虑，系统设计师不得不把一些低级或高级的功能综合起来。

（2）很难找到一个合适的、正确的层次抽象方法。

11.3.2 C2 风格

C2 架构风格可以概括为：通过连接件绑定在一起的按照一组规则运作的并行构件网络。C2 风格中的系统组织规则如下：

（1）系统中的构件和连接件都有一个顶部和一个底部；

（2）构件的顶部应连接到某连接件的底部，构件的底部则应连接到某连接件的顶部，而构件与构件之间的直接连接是不允许的；

（3）一个连接件可以和任意数目的其他构件和连接件连接；

（4）当两个连接件进行直接连接时，必须由其中一个的底部到另一个的顶部。

图 11-14 是 C2 风格的示意图。图中构件与连接件之间的连接体现了 C2 风格中构建系统的规则。

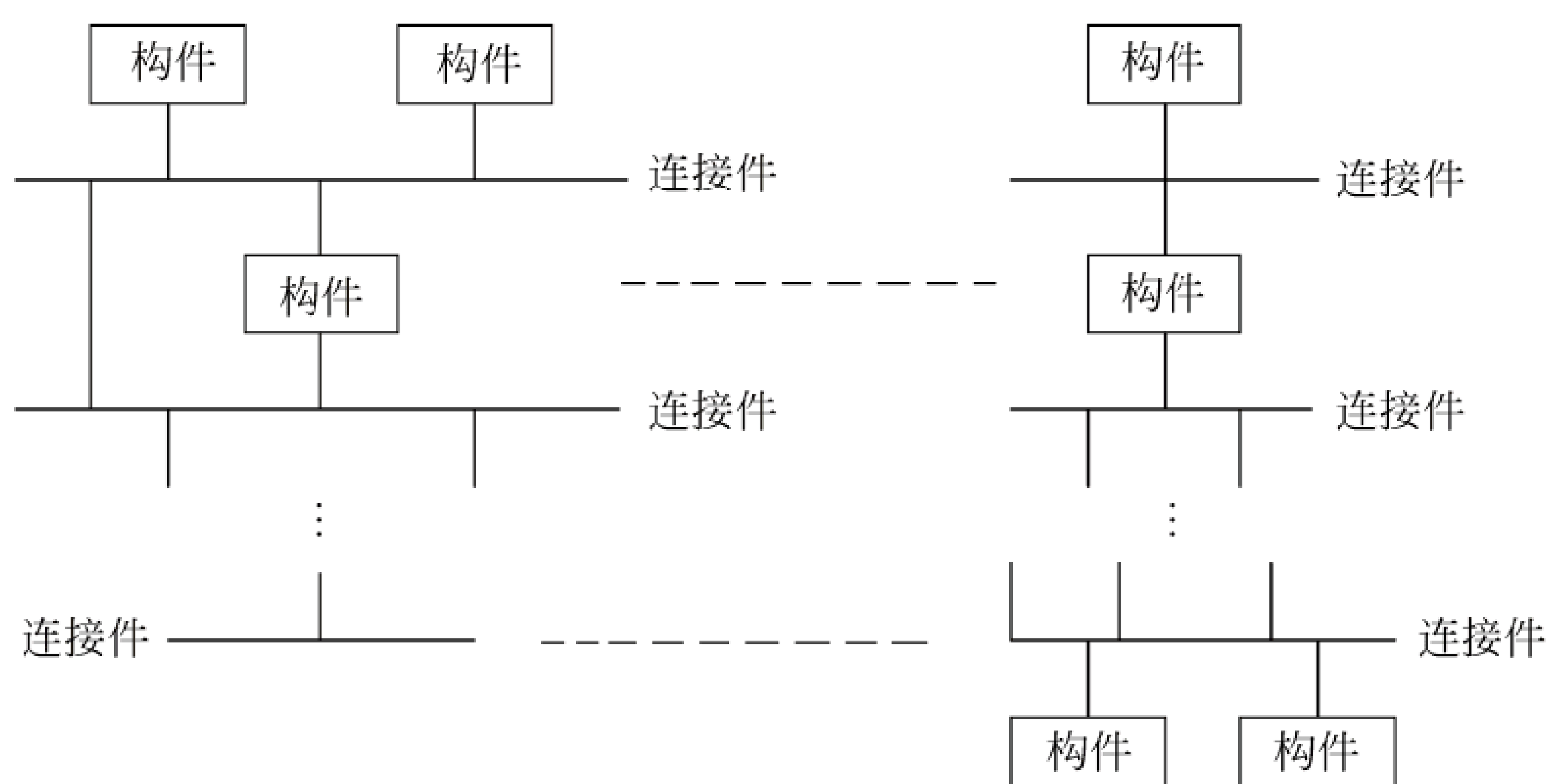


图 11-14　C2 风格的架构

C2 风格是最常用的一种软件架构风格。从 C2 风格的组织规则和结构图中，可以得

出，C2 风格具有以下特点：

（1）系统中的构件可实现应用需求，并能将任意复杂度的功能封装在一起；

（2）所有构件之间的通信是通过以连接件为中介的异步消息交换机制来实现的；

（3）构件相对独立，构件之间依赖性较少。系统中不存在某些构件将在同一地址空间内执行，或某些构件共享特定控制线程之类的相关性假设。

11.3.3　客户/服务器风格

客户/服务器（Client/Server，C/S）计算技术在信息产业中占有重要的地位。网络计算经历了从基于宿主机的计算模型到客户/服务器计算模型的演变。

在集中式计算技术时代广泛使用的是大型机/小型机计算模型。它是通过一台物理上与宿主机相连接的非智能终端来实现宿主机上的应用程序。在多用户环境中，宿主机应用程序即负责与用户的交互，又负责对数据的管理：宿主机上的应用程序一般也分为与用户交互的前端和管理数据的后端，即数据库管理系统。集中式的系统使用户能共享贵重的硬件设备，如磁盘机、打印机和调制解调器等。但随着用户的增多，对宿主机能力的要求很高，而且开发者必须为每个新的应用重新设计同样的数据管理构件。

20 世纪 80 年代以后，集中式结构逐渐被以 PC 为主的微机网络所取代。个人计算机和工作站的采用，永远改变了协作计算模型，从而导致了分散的个人计算模型的产生。一方面，由于大型机系统固有的缺陷，如缺乏灵活性，无法适应信息量急剧增长的需求，并为整个企业提供全面的解决方案等等。另一方面，由于微处理器的日新月异，其强大的处理能力和低廉的价格使微机网络迅速发展，已不仅仅是简单的个人系统，这便形成了计算机界的向下规模化（downsizing）。其主要优点是用户可以选择适合自己需要的工作站、操作系统和应用程序。

C/S 软件架构是基于资源不对等，且为实现共享而提出来的，是 20 世纪 90 年代成熟起来的技术，C/S 架构定义了工作站如何与服务器相连，以实现数据和应用分布到多个处理机上。C/S 架构有三个主要组成部分：数据库服务器、客户应用程序和网络。

C/S 架构将应用一分为二，服务器（后台）负责数据管理，客户机（前台）完成与用户的交互任务。服务器为多个客户应用程序管理数据，而客户程序发送、请求和分析从服务器接收的数据，这是一种“胖客户机（fat client）”，“瘦服务器（thin server）”的架构。其数据流图如图 11-15 所示。

在一个 C/S 架构的软件系统中，客户应用程序是针对一个小的、特定的数据集，如一个表的行来进行操作，而不是像文件服务器那样针对整个文件进行，对某一条记录进行封锁，而不是对整个文件进行封锁，因此保证了系统的并发性，并使网络上传输的数据量减到最少，从而改善了系统的性能。

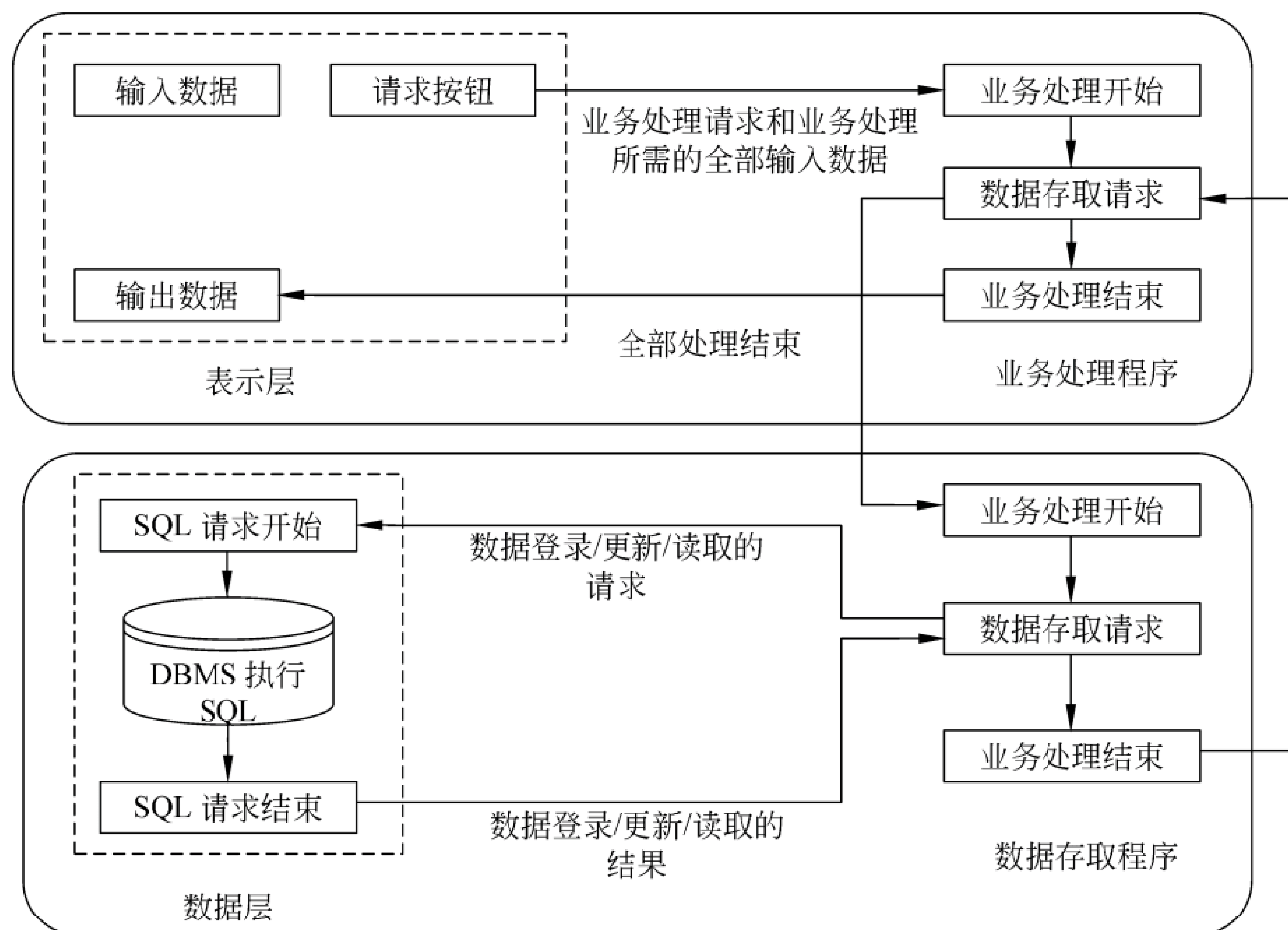


图 11-15　C/S 结构的一般处理流程

C/S 架构的优点主要在于系统的客户应用程序和服务器构件分别运行在不同的计算机上，系统中每台服务器都可以适合各构件的要求，这对于硬件和软件的变化显示出极大的适应性和灵活性，而且易于对系统进行扩充和缩小。在 C/S 架构中，系统中的功能构件充分隔离，客户应用程序的开发集中于数据的显示和分析，而数据库服务器的开发则集中于数据的管理，不必在每一个新的应用程序中都要对一个 DBMS 进行编码。将大的应用处理任务分布到许多通过网络连接的低成本计算机上，以节约大量费用。

C/S 架构具有强大的数据操作和事务处理能力，模型思想简单，易于人们理解和接受。但随着企业规模的日益扩大，软件的复杂程度不断提高，C/S 架构逐渐暴露了以下缺点：

（1）开发成本较高。C/S 架构对客户端软硬件配置要求较高，尤其是软件的不断升级，对硬件要求不断提高，增加了整个系统的成本，且客户端变得越来越臃肿。

（2）客户端程序设计复杂。采用 C/S 架构进行软件开发，大部分工作量放在客户端的程序设计上，客户端显得十分庞大。

（3）信息内容和形式单一，因为传统应用一般为事务处理，界面基本遵循数据库的字段解释，开发之初就已确定，而且不能随时截取办公信息和档案等外部信息，用户获得的只是单纯的字符和数字，既枯燥又死板。

（4）用户界面风格不一，使用繁杂，不利于推广使用。

（5）软件移植困难。采用不同开发工具或平台开发的软件，一般互不兼容，不能或很难移植到其他平台上运行。

（6）软件维护和升级困难。采用 C/S 架构的软件要升级，开发人员必须到现场为客户端升级，每个客户端上的软件都需维护。对软件的一个小小改动（例如只改动一个变量），每一个客户端都必须更新。

（7）新技术不能轻易应用。因为一个软件平台及开发工具一旦选定，不可能轻易更改。

11.3.4　三层 C/S 结构风格

传统的二层 C/S 结构存在以下几个局限：

（1）二层 C/S 结构是单一服务器且以局域网为中心的，所以难以扩展至大型企业广域网或 Internet。

（2）软、硬件的组合及集成能力有限。

（3）客户端的负荷太重，难以管理大量的客户端，系统的性能容易变坏。

（4）数据安全性不好。因为客户端程序可以直接访问数据库服务器，那么，在客户端计算机上的其他程序也可想办法访问数据库服务器，从而使数据库的安全性受到威胁。

正是因为二层 C/S 架构有这么多缺点，因此，三层 C/S 架构应运而生。与二层 C/S 结构相比，在三层 C/S 架构中，增加了一个应用服务器。可以将整个应用逻辑驻留在应用服务器上，而只有表示层存在于客户机上。这种结构被称为“瘦客户机”（Thin Client）。三层 C/S 架构是将应用功能分成表示层、功能层和数据层三个部分，如图 11-16 所示。

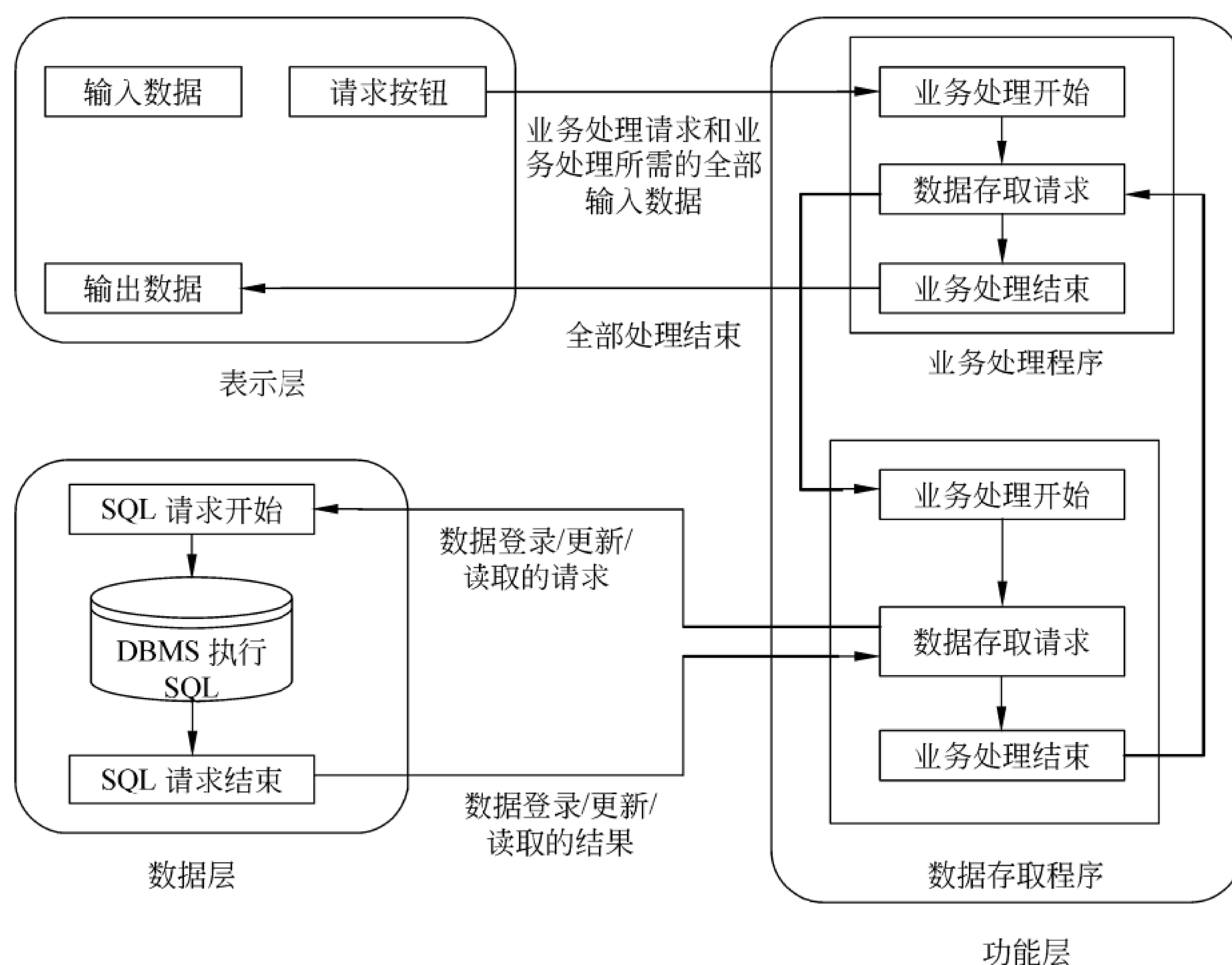


图 11-16　三层 C/S 结构的一般处理流程

1．表示层

表示层是应用的用户接口部分，它担负着用户与应用间的对话功能。它用于检查用户从键盘等输入的数据，显示应用输出的数据。为使用户能直观地进行操作，一般要使用图形用户界面，操作简单、易学易用。在变更用户界面时，只需改写显示控制和数据检查程序，而不影响其他两层。检查的内容也只限于数据的形式和取值的范围，不包括有关业务本身的处理逻辑。

2．功能层

功能层相当于应用的本体，它是将具体的业务处理逻辑编入程序中。例如，在制作订购合同时要计算合同金额，按照定好的格式配置数据、打印订购合同，而处理所需的数据则要从表示层或数据层取得。表示层和功能层之间的数据交往要尽可能简洁。例如，用户检索数据时，要设法将有关检索要求的信息一次性地传送给功能层，而由功能层处理过的检索结果数据也一次性地传送给表示层。

通常，在功能层中包含有确认用户对应用和数据库存取权限的功能以及记录系统处理日志的功能。功能层的程序多半是用可视化编程工具开发的，也有使用 COBOL 和 C 语言的。

3．数据层

数据层就是数据库管理系统，负责管理对数据库数据的读写。数据库管理系统必须能迅速执行大量数据的更新和检索。现在的主流是关系型数据库管理系统，因此，一般从功能层传送到数据层的要求大都使用 SQL 语言。

三层 C/S 的解决方案是：对这三层进行明确分割，并在逻辑上使其独立。原来的数据层作为数据库管理系统已经独立出来，所以，关键是要将表示层和功能层分离成各自独立的程序，并且还要使这两层间的接口简洁明了。

一般情况是只将表示层配置在客户机中，如图 11-17 所示。如果像图 11-17（3）所示的那样连功能层也放在客户机中，与二层 C/S 架构相比，其程序的可维护性要好得多，但是其他问题并未得到解决。客户机的负荷太重，其业务处理所需的数据要从服务器传给客户机，所以系统的性能容易变坏。

如果将功能层和数据层分别放在不同的服务器中，如图 11-17（2）所示，则服务器和服务器之间也要进行数据传送。但是，由于在这种形态中三层是分别放在各自不同的硬件系统上的，所以灵活性很高，能够适应客户机数目的增加和处理负荷的变动。例如，在追加新业务处理时，可以相应增加装载功能层的服务器。因此，系统规模越大这种形态的优点就越显著。

在三层 C/S 架构中，中间件是最重要的构件。所谓中间件是一个用 API 定义的软件层，是具有强大通信能力和良好可扩展性的分布式软件管理框架。它的功能是在客户机和服务器或服务器和服务器之间传送数据，实现客户机群和服务器群之间的通信。其工作流程是：在客户机里的应用程序需要驻留网络上某个服务器的数据或服务时，搜索此

数据的 C/S 应用程序需访问中间件系统。该系统将查找数据源或服务，并在发送应用程序请求后重新打包响应，将其传送回应用程序。

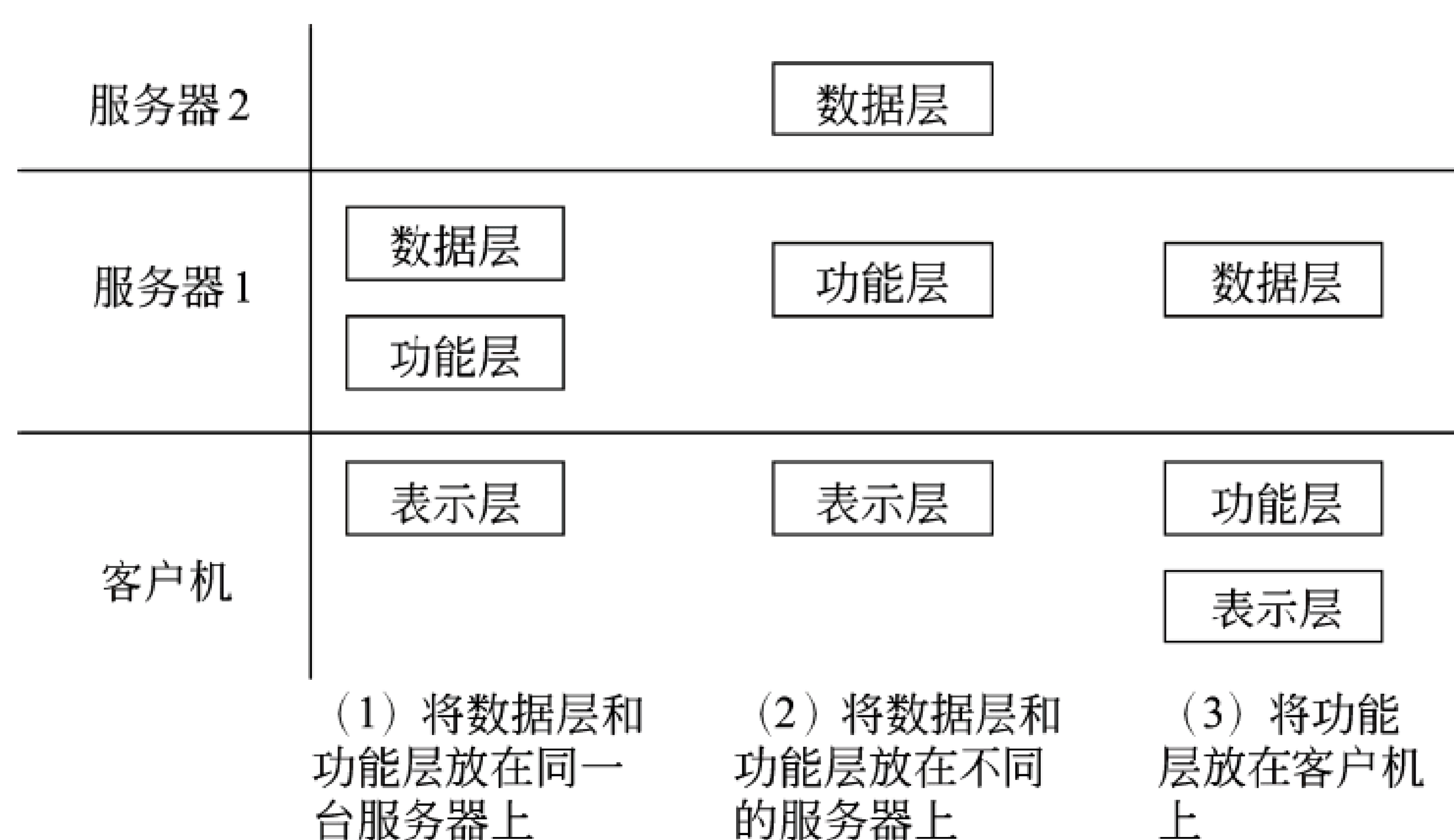


图 11-17　三层 C/S 物理结构比较

根据三层 C/S 的概念及使用实例，可以看出，与传统的二层结构相比，三层 C/S 结构具有以下优点。

（1）允许合理地划分三层结构的功能，使之在逻辑上保持相对独立性，从而使整个系统的逻辑结构更为清晰，能提高系统和软件的可维护性和可扩展性。

（2）允许更灵活有效地选用相应的平台和硬件系统，使之在处理负荷能力上与处理特性上分别适应于结构清晰的三层；并且这些平台和各个组成部分可以具有良好的可升级性和开放性。例如，最初用一台 UNIX 工作站作为服务器，将数据层和功能层都配置在这台服务器上。随着业务的发展，用户数和数据量逐渐增加，这时，就可以将 UNIX 工作站作为功能层的专用服务器，另外追加一台专用于数据层的服务器。若业务进一步扩大，用户数进一步增加，则可以继续增加功能层的服务器数目，用于分割数据库。清晰、合理地分割三层结构并使其独立，可以使系统构成的变更非常简单。因此，被分成三层的应用基本上不需要修正。

（3）三层 C/S 结构中，应用的各层可以并行开发，各层也可以选择各自最适合的开发语言。使之能并行地而且是高效地进行开发，达到较高的性能价格比；对每一层的处理逻辑的开发和维护也会更容易些。

（4）允许充分利用功能层有效地隔离开表示层与数据层，未授权的用户难以绕过功能层而利用数据库工具或黑客手段去非法地访问数据层，这就为严格的安全管理奠定了坚实的基础；整个系统的管理层次也更加合理和可控制。

希赛教育专家提示：三层 C/S 结构各层间的通信效率若不高，即使分配给各层的硬件能力很强，其作为整体来说也达不到所要求的性能。此外，设计时必须慎重考虑三层间的通信方法、通信频度及数据量。这和提高各层的独立性一样是三层 C/S 结构的关键

问题。

11.3.5 浏览器/服务器风格

在三层 C/S 架构中，表示层负责处理用户的输入和向客户的输出（出于效率的考虑，它可能在向上传输用户的输入前进行合法性验证）。功能层负责建立数据库的连接，根据用户的请求生成访问数据库的 SQL 语句，并把结果返回给客户端。数据层负责实际的数据库存储和检索，响应功能层的数据处理请求，并将结果返回给功能层。

浏览器/服务器（Browser/Server，B/S）风格就是上述三层应用结构的一种实现方式，其具体结构为：浏览器/Web 服务器/数据库服务器。B/S 架构主要是利用不断成熟的 WWW 浏览器技术，结合浏览器的多种脚本语言，用通用浏览器就实现了原来需要复杂的专用软件才能实现的强大功能，并节约了开发成本。从某种程度上来说，B/S 结构是一种全新的软件架构。

在 B/S 结构中，除了数据库服务器外，应用程序以网页形式存放于 Web 服务器上，用户运行某个应用程序时只须在客户端上的浏览器中输入相应的网址，调用 Web 服务器上的应用程序并对数据库进行操作完成相应的数据处理工作，最后将结果通过浏览器显示给用户。可以说，在 B/S 模式的计算机应用系统中，应用（程序）在一定程度上具有集中特征。

基于 B/S 架构的软件，系统安装、修改和维护全在服务器端解决。用户在使用系统时，仅仅需要一个浏览器就可运行全部的模块，真正达到了“零客户端”的功能，很容易在运行时自动升级。B/S 架构还提供了异种机、异种网、异种应用服务的联机、联网、统一服务的最现实的开放性基础。

B/S 结构出现之前，管理信息系统的功能覆盖范围主要是组织内部。B/S 结构的“零客户端”方式，使组织的供应商和客户（这些供应商和客户有可能是潜在的，也就是说可能是事先未知的！）的计算机方便地成为管理信息系统的客户端，进而在限定的功能范围内查询组织相关信息，完成与组织的各种业务往来的数据交换和处理工作，扩大了组织计算机应用系统的功能覆盖范围，可以更加充分利用网络上的各种资源，同时应用程序维护的工作量也大大减少。另外，B/S 结构的计算机应用系统与 Internet 的结合也使新近提出的一些新的企业计算机应用（如电子商务，客户关系管理）的实现成为可能。

与 C/S 架构相比，B/S 架构也有许多不足之处，例如：

（1）B/S 架构缺乏对动态页面的支持能力，没有集成有效的数据库处理功能。

（2）B/S 架构的系统扩展能力差。这是因为 B/S 架构的很多处理事务都在服务器上实现，一旦用户数量增多，就会导致服务器拥塞。

（3）B/S 架构的安全性难以控制。这是因为 B/S 架构的系统通常都是以网站形式发布在互联网上，除非采取特别措施，否则，所有的网民都可以访问。这样，随着各种破坏技术的发展，系统的安全性就会受到威胁。

（4）采用 B/S 架构的应用系统，在数据查询等响应速度上，要远远地低于 C/S 架构。

（5）B/S 架构的数据提交一般以页面为单位，数据的动态交互性不强，不利于在线事务处理应用。当然，随着富互联网应用架构技术的发展，这个问题会得到一定的缓解。详细情况，请阅读 11.6 节。

因此，虽然 B/S 结构的计算机应用系统有如此多的优越性，但由于 C/S 结构的成熟性且 C/S 结构的计算机应用系统网络负载较小，因此，未来一段时间内，将是 B/S 结构和 C/S 结构共存的情况。但是，很显然，计算机应用系统计算模式的发展趋势是向 B/S 结构转变。

11.3.6　公共对象请求代理架构

CORBA 是由 OMG 制定的一个工业标准，其主要目标是提供一种机制，使得对象可以透明地发出请求和获得应答，从而建立起一个异质的分布式应用环境。

由于分布式对象计算技术具有明显优势，OMG 提出了 CORBA 规范来适应该技术的进一步发展。1991 年，OMG 基于面向对象技术，给出了以 ORB 为中心的对象管理结构。

在 OMG 的对象管理结构中，ORB 是一个关键的通信机制，它以实现互操作性为主要目标，处理对象之间消息分布。对象服务实现基本的对象创建和管理功能，通用服务则使用对象管理结构所规定的类接口实现一些通用功能。

针对 ORB，OMG 又进一步提出了 CORBA 技术规范，主要内容包括接口定义语言（Interface Definition Language，IDL）、接口池（Interface Repository，IR）、动态调用接口（Dynamic Invocation Interface，DII）、对象适配器（Object Adapter，OA）等。

1．接口定义语言

CORBA 利用 IDL 统一地描述服务器对象（向调用者提供服务的对象）的接口。IDL 本身也是面向对象的。它虽然不是编程语言，但它为客户对象（发出服务请求的对象）提供了语言的独立性，因为客户对象只需了解服务器对象的 IDL 接口，不必知道其编程语言。IDL 语言是 CORBA 规范中定义的一种中性语言，它用来描述对象的接口，而不涉及对象的具体实现。在 CORBA 中定义了 IDL 语言到 C、C++、SmallTalk 和 Java 语言的映射。

2．接口池

CORBA 的接口池包括了分布计算环境中所有可用的服务器对象的接口表示。它使动态搜索可用服务器的接口、动态构造请求及参数成为可能。

3．动态调用接口

CORBA 的动态调用接口提供了一些标准函数以供客户对象动态创建请求、动态构造请求参数。客户对象将动态调用接口与接口池配合使用可实现服务器对象接口的动态搜索、请求及参数的动态构造与动态发送。当然，只要客户对象在编译之前能够确定服

务器对象的 IDL 接口，CORBA 也允许客户对象使用静态调用机制。显然，静态机制的灵活性虽不及动态机制，但执行效率却胜过动态机制。

4．对象适配器

在 CORBA 中，对象适配器用于屏蔽 ORB 内核的实现细节，为服务器对象的实现者提供抽象接口，以便他们使用 ORB 内部的某些功能。这些功能包括服务器对象的登录与激活、客户请求的认证等。

CORBA 定义了一种面向对象的软件构件构造方法，使不同的应用可以共享由此构造出来的软件构件。每个对象都将其内部操作细节封装起来，同时又向外界提供了精确定义的接口，从而降低了应用系统的复杂性，也降低了软件开发费用。CORBA 的平台无关性实现了对象的跨平台引用，开发人员可以在更大的范围内选择最实用的对象加入到自己的应用系统之中。CORBA 的语言无关性使开发人员可以在更大的范围内相互利用别人的编程技能和成果。

CORBA 的架构模式如图 11-18 所示。在此架构中，客户端应用程序用桩类型激发 API 或动态激发 API 向服务器发送请求。在服务器端接受方法调用请求，不进行参数引导，设置需要的上下文状态，激发服务器框架中的方法调度器，引导输出参数，并完成激发。服务器应用程序使用服务器端的服务部分，它包含了某个对象的一个或多个实现，用于满足客户机对指定对象上的某个操作的请求。很明显，客户机系统是独立于服务器系统的，同样，服务器系统也独立于客户机系统。

CORBA 架构模式充分利用了现今软件技术发展的最新成果，在基于网络的分布式应用环境下实现应用软件的集成，使得面向对象的软件在分布、异构环境下实现可重用、可移植和互操作。其特点可以总结为如下几个方面：

（1）引入中间件作为事务代理，完成客户机向服务对象方（server）提出的业务请求。

（2）实现客户与服务对象的完全分开，客户不需要了解服务对象的实现过程以及具体位置。

（3）提供软总线机制，使得在任何环境下、采用任何语言开发的软件只要符合接口规范的定义，均能够集成到分布式系统中。

（4）CORBA 规范软件系统采用面向对象的软件实现方法开发应用系统，实现对象内部细节的完整封装，保留对象方法的对外接口定义。

在以上特点中，最突出的是中间件的引入。对象模型是应用开发人员对客观事物属性和功能的具体抽象。由于 CORBA 使用了对象模型，将 CORBA 系统中所有的应用看成是对象及相关操作的集合，因此通过对象请求代理，使 CORBA 系统中分布在网络中应用对象的获取只取决于网络的畅通性和服务对象特征获取的准确程度，而与对象的位置以及对象所处的设备环境无关。

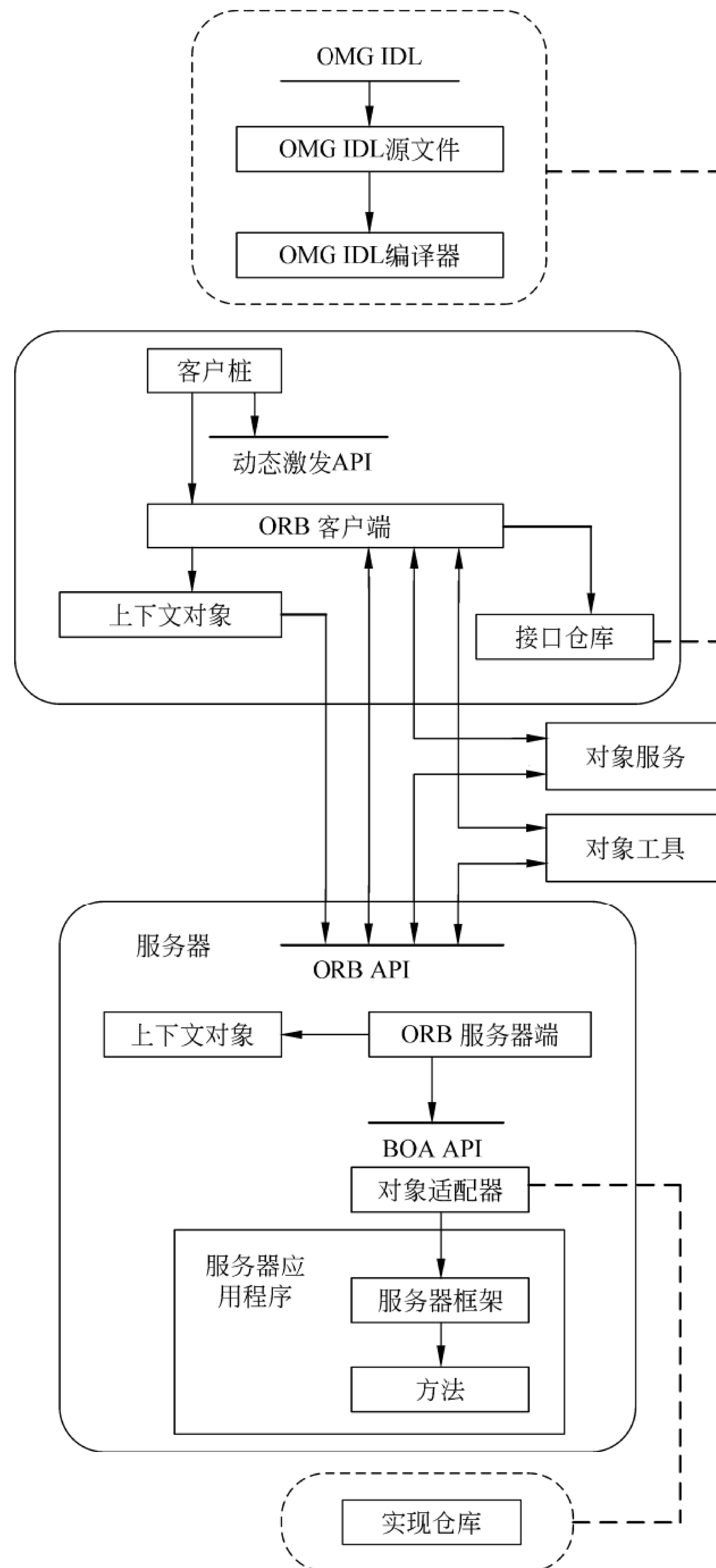


图 11-18　CORBA 的架构模式

11.3.7 异构结构风格

在前面的几节里，介绍和讨论了一些所谓的“纯”架构，但随着软件系统规模的扩大，系统也越来越复杂，所有的系统不可能都在单一的标准的结构上进行设计，这是因为：

（1）从最根本上来说，不同的结构有不同的处理能力的强项和弱点，一个系统的架构应该根据实际需要进行选择，以解决实际问题。

（2）关于软件包、框架、通信以及其他一些架构上的问题，目前存在多种标准。即使在某段时间内某一种标准占统治地位，但变动最终是绝对的。

（3）实际工作中，总会遇到一些遗留下来的代码，它们仍有效用，但是却与新系统有某种程度上的不协调。然而在许多场合，将技术与经济综合进行考虑时，总是决定不再重写它们。

（4）即使在某一单位中，规定了共享共同的软件包或相互关系的一些标准，仍会存在解释或表示习惯上的不同。在 UNIX 中就可以发现这类问题：即使规定用单一的标准（ASCII）来保证过滤器之间的通信，但因为不同人对关于在 ASCII 流中信息如何表示的不同的假设，不同的过滤器之间仍可能不协调。

软件架构风格为大粒度的软件重用提供了机会，然而，架构风格的使用却没有一个定式。对于应用架构风格来说，由于观察和考虑问题的角度不同，系统设计师可以有很大的选择余地。而且，不同的软件架构具有不同的处理能力的强项和弱点。因此，一个系统的架构应该根据实际需要进行选择，要为系统选择或设计某一个架构风格，必须根据特定项目的具体特点，进行分析和比较后才能确定。

事实上，也存在一些系统，它们是由这些纯架构组合而成，即采用了异构软件架构。异架构的组合方式有很多种，例如，可以采用平行的方式，即根据软件各个子系统的结构、功能和性能，为每个/类子系统选择相应的架构。例如，笔者在设计某省工商行政管理系统时，根据子系统的功能把所有子系统划分为两大类，分别是数据处理类和查询类。然后根据子系统的功能选择相应的架构，为数据处理类子系统选择了 C/S 结构，为查询类子系统选择了 B/S 结构。

11.4 特定领域软件架构

早在 20 世纪 70 年代就有人提出程序族、应用族的概念，并开拓了对特定领域软件架构早期研究，这与软件架构研究的主要目的“在一组相关的应用中共享软件架构”也是一致的。为了解脱因为缺乏可用的软件构件以及现有软件构件难以集成而导致软件开发过程中难以进行重用的困境，Mettala 在 1990 年提出了特定领域软件架构（Domain Specific Software Architecture，DSSA），尝试解决这类问题。

简单地说，DSSA 就是一个特定的问题领域中支持一组应用的领域模型、参考需求、参考架构等组成的开发基础，其目标就是支持在一个特定领域中多个应用的生成。

DSSA 的必备特征为：

（1）一个严格定义的问题域和/或解决域。

（2）具有普遍性，使其可以用于领域中某个特定应用的开发。

（3）对整个领域的合适程度的抽象。

（4）具备该领域固定的、典型的在开发过程中可重用元素。

从功能覆盖的范围角度有两种理解 DSSA 中领域的含义的方式。

（1）垂直域：定义了一个特定的系统族，包含整个系统族内的多个系统，结果是在该领域中可作为系统的可行解决方案的一个通用软件架构。

（2）水平域：定义了在多个系统和多个系统族中功能区域的共有部分，在子系统级上涵盖多个系统族的特定部分功能，无法为系统提供完整的通用架构。

在垂直域上定义的 DSSA 只能应用于一个成熟的、稳定的领域，但这个条件比较难以满足；若将领域分割成较小的范围，则更相对容易，也容易得到一个一致的解决方案。

11.4.1　DSSA 的活动

DSSA 的主要活动可以分为三个阶段，分别是领域分析、领域设计和领域实现。

1．领域分析

这个阶段的主要目标是获得领域模型（Domain Model）。领域模型描述领域中系统之间的共同的需求。称领域模型所描述的需求为领域需求。在这个阶段中，首先要进行一些准备性的活动，包括定义领域的边界，从而明确分析的对象；识别信息源，即领域分析和整个领域工程过程中信息的来源，可能的信息源包括现存系统、技术文献、问题域和系统开发的专家、用户调查和市场分析、领域演化的历史记录等。在此基础上，就可以分析领域中系统的需求，确定哪些需求是被领域中的系统广泛共享的，从而建立领域模型。当领域中存在大量系统时，需要选择它们的一个子集作为样本系统。对样本系统需求的考察将显示领域需求的一个变化范围。一些需求对所有被考察的系统是共同的，一些需求是单个系统所独有的。很多需求位于这两个极端之间，即被部分系统共享。

2．领域设计

这个阶段的目标是获得 DSSA。DSSA 描述在领域模型中表示的需求的解决方案，它不是单个系统的表示，而是能够适应领域中多个系统的需求的一个高层次的设计。建立了领域模型之后，就可以派生出满足这些被建模的领域需求的 DSSA。由于领域模型中的领域需求具有一定的变化性，DSSA 也要相应地具有变化性。它可以通过表示多选一的（alternative）、可选的（optional）解决方案等来做到这一点。由于重用基础设施是依据领域模型和 DSSA 来组织的，因此在这个阶段通过获得 DSSA，也就同时形成了重用基础设施的规约。

3．领域实现

这个阶段的主要目标是依据领域模型和 DSSA 开发和组织可重用信息。这些可重用信息可能是从现有系统中提取得到，也可能需要通过新的开发得到。它们依据领域模型和 DSSA 进行组织，也就是领域模型和 DSSA 定义了这些可重用信息的重用时机，从而支持了系统化的软件重用。这个阶段也可以看作重用基础设施的实现阶段。

希赛教育专家提示：以上过程是一个反复的、逐渐求精的过程。在实施领域工程的每个阶段中，都可能返回到以前的步骤，对以前的步骤得到的结果进行修改和完善，再回到当前步骤，在新的基础上进行本阶段的活动。

11.4.2 DSSA 的建立过程

因所在的领域不同，DSSA 的创建和使用过程也各有差异，Tracz 曾提出了一个通用的 DSSA 应用过程，这些过程也需要根据所应用到的领域来进行调整。一般情况下，需要用所应用领域的应用开发者习惯使用的工具和方法来建立 DSSA 模型。同时，Tracz 强调了 DSSA 参考架构文档工作的重要性，因为新应用的开发和对现有应用的维护都要以此为基础。

DSSA 的建立过程分为 5 个阶段，每个阶段可以进一步划分为一些步骤或子阶段。每个阶段包括一组需要回答的问题，一组需要的输入，一组将产生的输出和验证标准。该过程是并发的（concurrent）、递归的（recursive）、反复的（iterative），或可以说，它是螺旋型（spiral）的。完成该过程可能需要对每个阶段经历几遍，每次增加更多的细节。

（1）定义领域范围：本阶段的重点是确定什么在感兴趣的领域中以及本过程到何时结束。这个阶段的一个主要输出是领域中的应用需要满足一系列用户的需求。

（2）定义领域特定的元素：本阶段的目标是编译领域字典和领域术语的同义词词典。在领域工程过程的前一个阶段产生的高层块图将被增加更多的细节，特别是识别领域中应用间的共同性和差异性。

（3）定义领域特定的设计和实现需求约束：本阶段的目标是描述解空间中有差别的特性。不仅要识别出约束，并且要记录约束对设计和实现决定造成的后果，还要记录对处理这些问题时产生的所有问题的讨论。

（4）定义领域模型和架构：本阶段的目标是产生一般的架构，并说明构成它们的模块或构件的语法和语义。

（5）产生、搜集可重用的产品单元：本阶段的目标是为 DSSA 增加构件使得它可以被用来产生问题域中的新应用。

DSSA 的建立过程是并发的、递归的和反复进行的。该过程的目的是将用户的需要映射为基于实现限制集合的软件需求，这些需求定义了 DSSA。在此之前的领域工程和领域分析过程并没有对系统的功能性需求和实现限制进行区分，而是统称为“需求”。图 11-19 是 DSSA 的一个三层次系统模型。

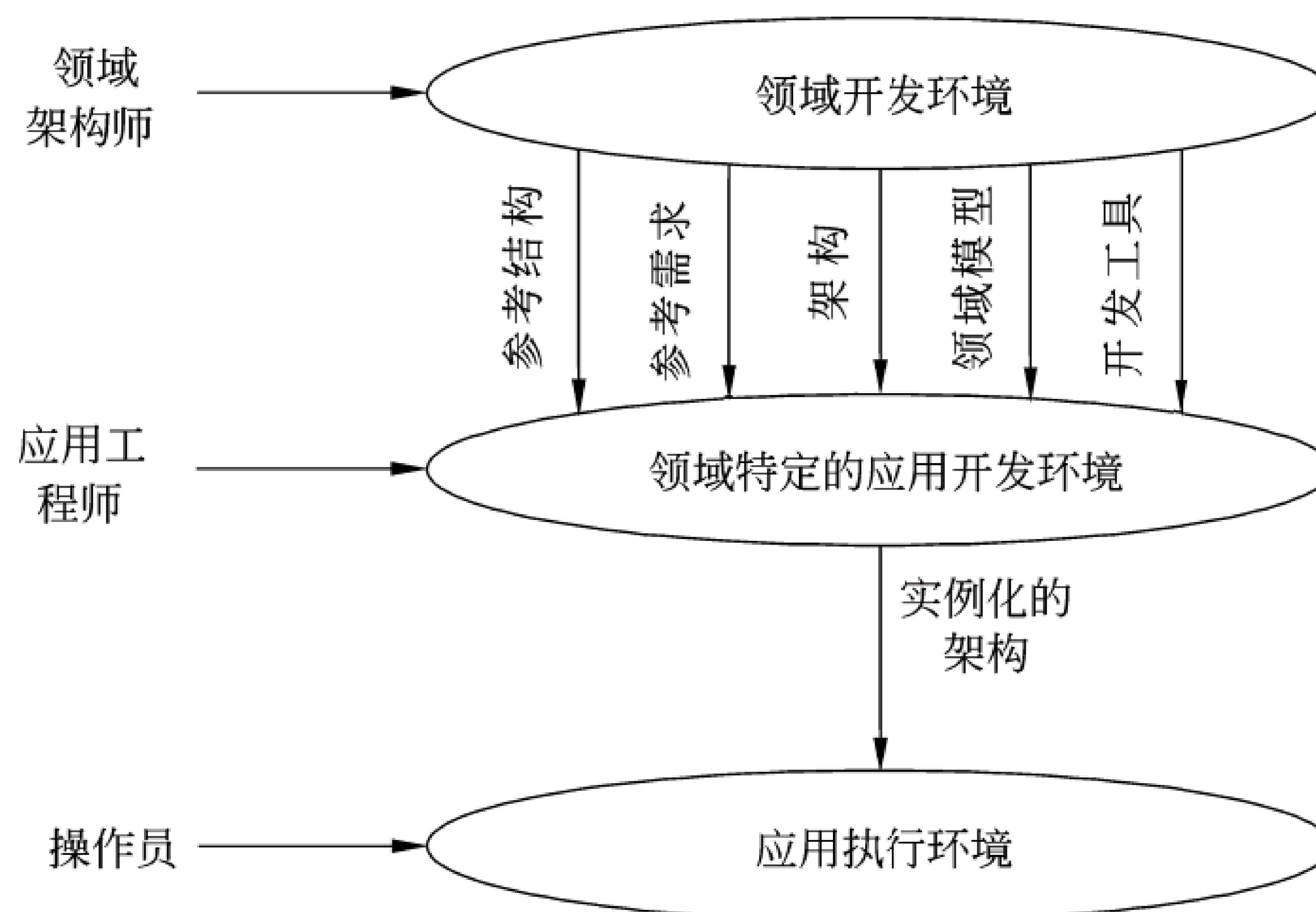


图 11-19　DSSA 的一个三层次系统模型

DSSA 的建立需要设计人员对所在特定应用领域（包括问题域和解决域）必须精通，他们要找到合适的抽象方式来实现 DSSA 的通用性和可重用性。通常，DSSA 以一种逐渐演化的方式发展。

11.5　面向服务的架构

面向服务的架构（Service Oriented Architecture，SOA）是一种新的架构风格，它具有松耦合和面向软件服务的特点，具有很高的重用性和灵活性。而要实现 SOA 的这些优点，Web Service 在其中扮演了重要的角色。

11.5.1　SOA 的概念

由于 SOA 还是一个比较新的概念，不同厂商对 SOA 的定义也不一样，下面，列举几个主要的定义。

W3C：SOA 是指服务提供者完成一组工作，为服务使用者交付所需的最终结果。最终结果通常会使使用者的状态发生变化，但也可能使提供者的状态改变，或双方都产生变化。

Service-architecture.com：SOA 本质上是服务的集合，服务间彼此通信，这种通信可能是简单的数据传送，也可能是两个或更多的服务协作进行某些活动。服务间需要某些方法进行连接。所谓服务就是精确定义、封装完善、独立于其他服务所处环境和状态的函数。

Looselycoupled.com：SOA 就是按需连接资源的系统。在 SOA 中，资源被作为可通过标准方式访问的独立服务，提供给网络中的其他成员。与传统的系统架构相比，SOA

规定了资源间更为灵活的松散耦合关系。

Gartner：SOA 是一种 C/S 模式的软件设计方法，在 SOA 中，一项应用由软件服务和软件服务使用者组成。SOA 与大多数通用的 C/S 模型的不同之处，在于它着重强调软件构件的松散耦合，并使用独立的标准接口。

META：SOA 是一种以通用为目的、可扩展、具有联合协作性的架构，所有流程都被定义为服务，服务通过基于类封装的服务接口委托给服务提供者，服务接口根据可扩展标识符、格式和协议单独描述。

从上述不同的定义中，可以得出 SOA 的几个关键特性：一种粗粒度、松耦合服务架构，服务之间通过简单、精确定义接口进行通信，不涉及底层编程接口和通信模型。

在 SOA 中，服务是封装成用于业务流程的可重用构件的 API。它提供信息或简化业务数据从一个有效的、一致的状态向另一个状态的转变。用于实现特定服务的流程并不重要，只要它响应用户的命令并为用户的请求提供高质量的服务就可以了。

通过定义的通信协议，可以调用服务来强调互操作性和位置透明性。一个服务表现为一个软件构件，因为从服务请求者的角度来看，它看起来就像是一个自包含的函数。然而，实际上，服务的实现可能包括在一个企业内部的不同计算机上或许多业务合作伙伴拥有的计算机上执行的很多步骤。就封装的软件而言，服务可能是一个构件，也可能不是一个构件。如同类对象，请求者应用程序能够将服务看作是一个整体。

希赛教育专家提示：SOA 并不是一种现成的技术，而是一种架构和组织 IT 基础结构及业务功能的方法。SOA 是一种在计算环境中设计、开发、部署和管理离散逻辑单元（服务）的模型。

SOA 要求开发人员将应用设计为服务的集合，要求开发人员跳出应用本身进行思考，考虑现有服务的重用，或思索他们的服务如何能够被其他项目重用。但是，SOA 并不仅仅是一种开发方法，它还具有管理上的优点。例如，现在管理员可直接管理开发人员所构建的相同服务，这远胜于以往管理单个应用的方式。通过分析服务间的交互，SOA 可以帮助企业了解何时以及为什么业务逻辑被切实执行了，这使管理员或分析师能够有针对性地优化业务流程。

11.5.2 SOA 的特征

实施 SOA 的关键目标是实现企业 IT 资产重用的最大化。要实现这一目标，就要在实施 SOA 的过程中牢记以下特征。

1．可从企业外部访问

商业合作伙伴（外部用户）也能像企业内部用户一样访问相同的服务，商业合作伙伴采用 B2B（Business To Business）协议（如 ebXML、RosettaNet 等）相互合作。当商业合作伙伴基于业务目的交换业务信息时，他们就参与了一次会话。会话是商业合作伙伴之间一系列的一条或多条业务信息的交换。会话类型（会话复杂或简单、长或短等）

取决于业务目的。

除了 B2B 协议外，外部用户还可以访问以 Web 服务方式提供的企业服务。

2．随时可用

当有服务使用者请求服务时，SOA 要求必须有服务提供者能够响应。大多数 SOA 都能够为门户应用之类的同步应用和 B2B 之类的异步应用提供服务。同步应用对于其所使用的服务具有很强的依赖性。

许多同步应用通常部署在前台，其最终用户很容易受到服务提供者短缺的影响。很多情况下，同步应用利用分布式服务提供者，这样可以响应更多的用户请求。但是，随着提供特定服务功能的服务器数量的增长，出现短缺的可能性也呈指数级上升。

相比之下，异步应用要更为稳健，因为其采用队列请求设计，因此，可以容许出现服务提供者短缺或迟滞的情况。异步应用大多数情况下部署在后台，用户通常不会觉察到短暂的短缺。大部分情况下异步应用能够稳健应对短时间短缺，但是长时间短缺则会引发严重问题。在服务短缺解决、队列引擎将罕见的大量工作推到共享的应用资源中时，可能会出现队列溢出甚至服务死锁。

服务使用者要求提供同步服务时，通常是基于其自身理解或使用习惯。在多数情况下，采用异步模型可以达到同样的效果，但更能够体现 SOA 的最佳特性。

当然，并不是所有情况下都应当采用异步设计模式。但大多数情况下，异步消息可以确保系统在不同负荷下的伸缩性，在接口响应时间不是很短时尤其如此。

3．粗粒度服务接口

粗粒度服务提供一项特定的业务功能，而细粒度服务代表了技术构件方法。例如，向希赛教育的视频点播系统中添加一个客户是典型的粗粒度服务，而用户可以使用几个细粒度服务实现同一功能，例如，将客户名加入到点播系统中、添加详细的客户联系方式、添加点播信息等。

采用粗粒度服务接口的优点在于使用者和服务层之间不必再进行多次的往复，一次往复就足够。Internet 环境中有保障的 TCP/IP 会话已不再占据主导、建立连接的成本也过高，因此，在该环境中进行应用开发时粗粒度服务接口的优点更为明显。

除去基本的往复效率，事务稳定性问题也很重要。在一个单独事务中包含的多段细粒度请求可能使事务处理时间过长、导致后台服务超时，从而中止。与此相反，从事务的角度来看，向后台服务请求大块数据可能是获取反馈的唯一途径。

4．分级

一个关于粗粒度服务的争论是此类服务比细粒度服务的重用性差，因为粗粒度服务倾向于解决专门的业务问题，因此，通用性差、重用性设计困难。解决该争论的方法之一就是允许采用不同的粗粒度等级来创建服务。这种服务分级包含了粒度较细、重用性较高的服务，也包含粒度较粗、重用性较差的服务。

在服务分级方面，必须注意服务层的公开服务通常由后台系统或 SOA 平台中现有

的本地服务组成。因此，允许在服务层创建私有服务是非常重要的。正确的文档、配置管理和私有服务的重用对于 IT 部门在 SOA 服务层快速开发新的公开服务的能力具有重要影响。

5．松散耦合

SOA 具有松散耦合构件服务，这一点区别于大多数其他的架构。该方法旨在将服务使用者和服务提供者在服务实现和客户如何使用服务方面隔离开来。

服务提供者和服务使用者间松散耦合背后的关键点是，服务接口作为与服务实现分离的实体而存在，这使得服务实现能够在完全不影响服务使用者的情况下进行修改。

大多数松散耦合方法都依靠基于服务接口的消息。基于消息的接口能够兼容多种传输方式（如 HTTP、JMS、TCP/IP、MOM 等）。基于消息的接口可以采用同步和异步协议实现，Web 服务对于 SOA 服务接口来讲是一个重要的标准。

当使用者调用一个 Web 服务时，被调用的对象可以是 CICS 事务、DCOM 或 CORBA 对象、J2EE EJB 或 Tuxedo 服务等，但这与服务使用者无关，底层实现并不重要。

6．可重用的服务及服务接口设计管理

如果完全按照可重用的原则设计服务，SOA 将可以使应用变得更为灵活。可重用服务采用通用格式提供重要的业务功能，为开发人员节约了大量时间。设计可重用服务是与数据库设计或通用数据建模类似的最有价值的工作。由于服务设计是成功的关键，因此 SOA 实施者应当寻找一种适当的方法进行服务设计过程管理。

服务设计管理从根本上讲是服务设计问题，服务设计需要在两点间折中，走捷径的项目战术与企业构建可重用通用服务的长期目标。

超越项目短期目标进行服务接口的开发和评估是迈向精确定义服务接口的重要一步，同时还需要为接口文档、服务实现文档及所有重要的非功能性特征设立标准。

在大型组织中实现重用的一个先决条件是建立通用（设计阶段）服务库和开发流程，以保证重用的正确性和通用性。此外，对记述服务设计和开发的服务文档进行评估也是成功利用服务库的关键。

简言之，不按规则编写服务将无法保证可提供重用性的 SOA 的成功实施。在执行规则的过程中会产生财务费用，需要在制定 SOA 实施计划时加以考虑。

7．标准化的接口

XML 和 Web Service 将 SOA 推向更高的层面，并大大提升了 SOA 的价值。尽管以往的 SOA 产品都是专有的，并且要求 IT 部门在其特定环境中开发所有应用，但 XML 和 Web Service 标准化的开放性使企业能够在所部署的所有技术和应用中采用 SOA。

Web Service 使应用功能得以通过标准化接口（WSDL）提供，并可基于标准化传输方式（HTTP 和 JMS）、采用标准化协议（SOAP）进行调用。例如，开发人员可以采用最适于门户开发的工具轻松创建一个新的门户应用，并可以重用 ERP 系统和定制化 J2EE 应用中的现有服务，而完全无须了解这些应用的内部工作原理。采用 XML，门户开发人

员无须了解特定的数据表示格式，便能够在这些应用间轻松地交换数据。

当然，用户也可以不采用 Web 服务或 XML 来创建 SOA 应用，但是，这两种标准的重要性日益增加、应用日趋普遍，大多数的服务使用者都会将其作为企业的服务访问方法。

8．支持各种消息模式

SOA 中可能存在以下消息模式。在一个 SOA 实现中，常会出现混合采用不同消息模式的服务。

（1）无状态的消息。服务使用者向服务提供者发送的每条消息都必须包含服务提供者处理该消息所需的全部信息。这一限定使服务提供者无须存储服务使用者的状态信息，从而更易扩展。

（2）有状态的消息。服务使用者与服务提供者共享服务使用者的特定环境信息，此信息包含在服务提供者和服务使用者交换的消息中。这一限定使服务提供者与服务使用者之间的通信更加灵活，但由于服务提供者必须存储每个服务使用者的共享环境信息，因此其整体可扩展性明显减弱。该限定增强了服务提供者和服务使用者的耦合关系，提高了交换服务提供者的服务难度。

（3）等幂消息。向软件代理发送多次重复消息的效果和发送单条消息相同。这一限定使服务提供者和服务使用者能够在出现故障时简单的复制消息，从而改进服务可靠性。

9．精确定义的服务接口

服务是由提供者和使用者间的契约定义的，契约规定了服务使用方法及使用者期望的最终结果。此外，还可以在其中规定服务质量。此处需要注意的关键点是，服务契约必须进行精确定义。

11.5.3　SOA 的优点和缺点

SOA 业务流程是由一系列业务服务组成的，可以更轻松地创建、修改和管理它来满足不同时期的需要。了解了 SOA 的定义和基本特征后，再来看看 SOA 潜在的优点。

（1）编码灵活性。可基于模块化的低层服务、采用不同组合方式创建高层服务，从而实现重用，这些都体现了编码的灵活性。此外，由于服务使用者不直接访问服务提供者，这种服务实现方式本身也可以灵活使用。

（2）明确开发人员角色。例如，熟悉黑莓企业版服务器（Black Berry Enterprise Server，BES）的开发人员可以集中精力在重用访问层，协调层开发人员则无须特别了解 BES 的实现，而将精力放在解决高价值的业务问题上。

（3）支持多种客户类型。借助精确定义的服务接口和对 XML、Web 服务标准的支持，可以支持多种客户类型，包括个人数码助理（Personal Digital Assistant，PDA）、手机等新型访问渠道。

（4）更易维护。服务提供者和服务使用者的松散耦合关系及对开放标准的采用确保

了该特性的实现。

（5）更好的伸缩性。依靠服务设计、开发和部署所采用的架构模型实现伸缩性。服务提供者可以彼此独立调整，以满足服务需求。

（6）更高的可用性。该特性在服务提供者和服务使用者的松散耦合关系上得以体现，服务使用者无须了解服务提供者的实现细节，这样，服务提供者就可以在集群环境中灵活部署，服务使用者可以被转接到可用的例程上。

（7）利用现有的资产。方法是将这些现有的资产包装成提供企业功能的服务。组织可以继续从现有的资源中获取价值，而不必重新从头开始构建。

（8）更易于集成和管理复杂性。将基础设施和实现发生的改变所带来的影响降到最低限度。因为复杂性是隔离的，当更多的企业一起协作提供价值链时，这会变得更加重要。

（9）更快地整合现实。通过利用现有的构件和服务，可以减少完成软件开发生命周期所需的时间。这使得可以快速地开发新的业务服务，并允许组织迅速地对改变做出响应和缩短开发时间。

（10）减少成本和增加重用。通过以松散耦合的方式公开业务服务，企业可以根据业务要求更轻松地使用和组合服务。

作为一个具有发展前景的应用系统架构，SOA 尚处在不断的发展中，存在很多有待改进的地方。Stencil Group 公司的 Brent Sleeper 指出，SOA 在可靠性、安全性、编制、遗留系统支持、语义和性能方面还存在严重不足。

（1）可靠性。SOA 还没有完全为事务的最高可靠性——不可否认性（nonrepudiation）、消息一定会被传送且仅传送一次（Once And Only Once Delivery）以及事务撤回（rollback）——做好准备。

（2）安全性。在过去，访问控制只需要登录和验证，而在 SOA 环境中，由于一个应用软件的构件很容易去跟属于不同域的其他构件进行对话，所以确保迥然不同又相互连接的系统之间的安全性就复杂得多。

（3）编排（orchestration）。统一协调分布式软件构件以便构建有意义的业务流程是最复杂的，但它同时也最适合面向服务类型的集成，原因很显然，建立在 SOA 上面的应用软件可以被设计成可以按需要拆散、重新组装的服务。作为目前业务流程管理解决方案的核心，编排功能使 IT 管理人员能够通过已经部署的套装或自己开发的应用软件的功能，把新的元应用软件（Meta Application Software，MAS）连接起来。事实上，最大的难题不是建立模块化的应用软件，而是改变这些系统表示所处理数据的方法。

（4）遗留系统支持。SOA 中提供集成遗留系统的适配器，遗留应用适配器屏蔽了许多专用性 API 的复杂性和晦涩性。一个设计良好的适配器的作用好比是一个设计良好的 SOA 服务，它提供了一个抽象层，把应用基础设施的其余部分与各种棘手问题隔离开来。一些厂商就专门把遗留应用软件“语义集成”到基于 XML 的集成架构中。但是，集成

遗留系统的工作始终是一个挑战。

（5）语义。定义事务和数据的业务含义，一直是 IT 管理人员面临的最棘手问题。语义关系是设计良好 SOA 架构的核心要素。就目前而言，没有哪一项技术或软件产品能够真正解决语义问题。为针对特定行业和功能的流程定义并实施功能和数据模型是一项繁重的任务，它最终必须由业务和 IT 管理人员共同承担。不过，预制构件和经过实践证明的咨询技能可以简化许多难题。

（6）性能。这种怀疑观点通常针对两个方面：SOA 的分布性质和 Web 服务协议的开销。不可否认，任何分布式系统的执行速度都不如独立式系统，这完全是因为网络的制约作用造成的。当然，有些应用软件无法容忍网络引起的延迟（例如，那些对实时性要求很高的应用软件），所以在应用 SOA 架构之前，搞清楚它的适用范围就显得很重要了。

11.5.4　SOA 的生命周期

由于 SOA 涉及到业务的诸多方面，因此需要从一开始就对 SOA 项目进行细心的规划和设计。需要考虑项目的整个生命周期，从最初的阶段到第一个实现，再一直到可能的修订和重用。SOA 生命周期如图 11-20 所示。

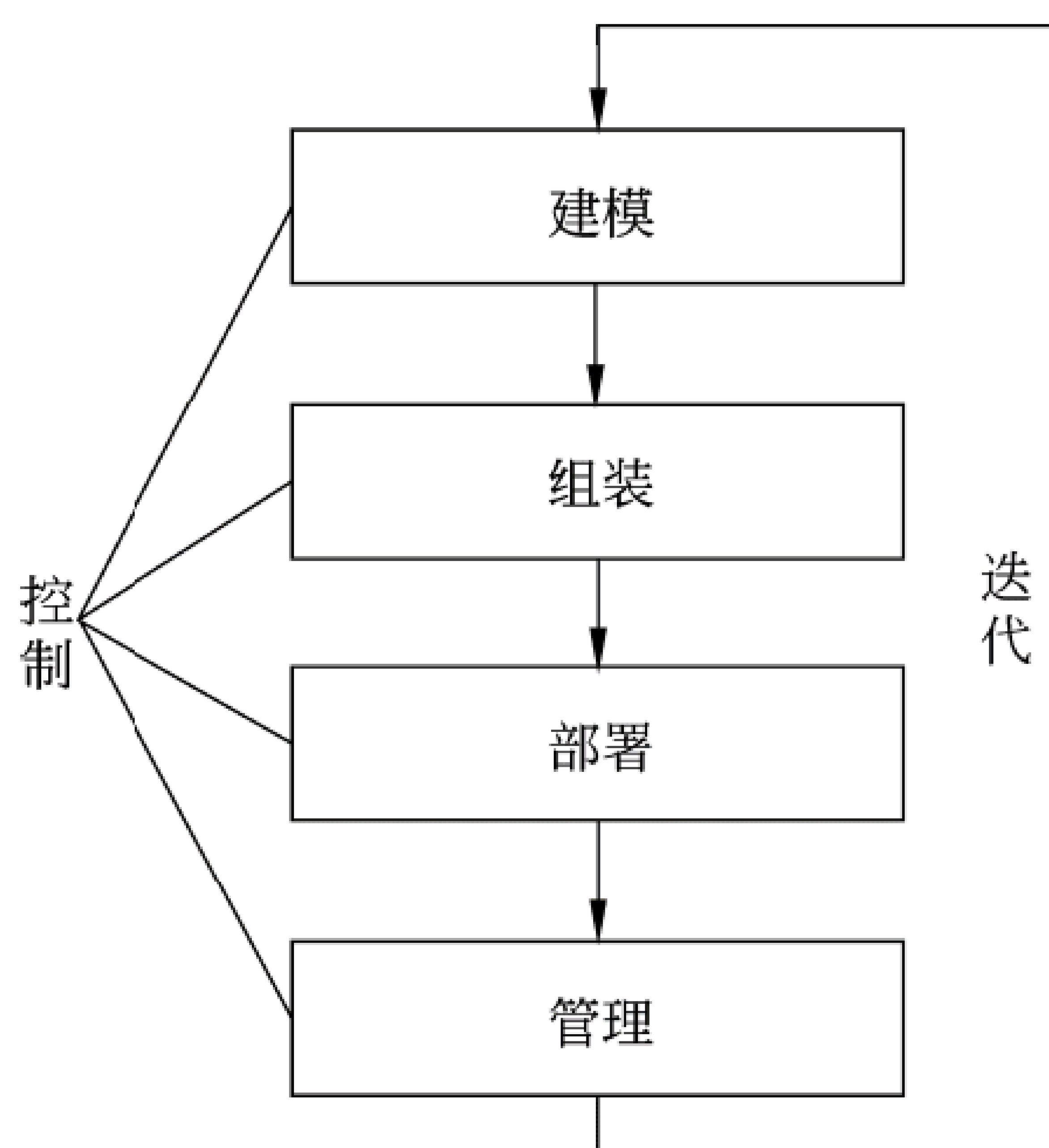


图 11-20　SOA 生命周期

1．建模

SOA 项目的第一步几乎和技术没有任何关系，所有事项都与用户的业务相关。面向服务的方法将业务所执行的活动视为服务，因此第一步是要确定这些业务活动或流程实际是什么。需要对业务架构进行记录，这些记录不仅可以用于规划 SOA，还可以用于对实际业务流程进行优化。通过在编写代码前模拟或建模业务流程，可以更深入地了解这

些流程，从而有利于构建帮助执行这些流程的软件。

建模业务流程的程度将依赖于预期实现的深度。另外，这个程度还依赖于开发人员在开发团队中担任的角色。如果是企业架构师，将会对实际的业务服务进行建模。如果是软件开发人员，将可能对单个服务进行建模。

2．组装

对业务流程进行了建模和优化后，开发人员可以开始构建新的服务和/或重用现有的服务，然后对其进行组装以形成组合应用程序，从而实现这些流程。在建模过程中，已经确定了需要何种类型的服务，以及它们将访问何种类型的数据，已经存在某种形式的实现这些服务或访问该类数据所需的一些软件。

组装过程将要找到已经存在的功能，并为其添加服务支持。另外，还涉及到创建提供功能和访问数据源所需的新服务，以便满足 SOA 涉及的业务流程范围内的需求。

3．部署

进行建模和组装后，要将组成 SOA 的资产部署到安全的集成环境中。此环境本身提供专门化的服务，用于集成业务中涉及的人员、流程和信息。这种级别的集成可帮助确保将公司的所有主要元素连接到一起协同工作。此外，部署工作还需要满足业务的性能和可用性需求，并提供足够的灵活性，以便吸纳新服务（并使旧服务退役），而不会对整个系统造成大的影响。

4．管理

部署之后，还需要从 IT 和业务两个角度对系统进行管理和监视。在管理过程中收集的信息用于帮助实时地了解业务流程，从而能更好地进行业务决策，并将信息反馈回生命周期，以进行持续的流程改进工作。将需要处理服务质量、安全、一般系统管理之类的问题。

在管理过程中，可以监视和优化系统，发现和纠正效率低下的情况和存在的问题。由于 SOA 是一个迭代过程，因此，在管理过程中，不仅要找出技术架构中有待改进的地方，而且还要找出业务架构中有待改进的地方。

完成管理过程后，就要开始新的建模过程了。在管理过程中收集的数据将用于重复整个 SOA 生命周期，再次进行整个过程。

5．控制

SOA 可以包含来自组织的不同部门的服务，甚至还可能包含来自组织外的服务。如果没有恰当的控制，这种系统很容易失控。

控制对所有生命周期阶段起到巩固支撑作用，为整个 SOA 系统提供指导，并有助于了解系统全貌。它提供指导和控制，帮助服务提供者和使用者避免遇到意外情况。

11.5.5　SOA 与其他技术的关系

SOA 可以与许多其他技术结合在一起使用，然而，构件的封装和聚合在其中扮演着重要的角色。

1．SOA 与编程语言

除了可能离不开 XML 和 WSDL 之外，SOA 并不是特定于语言的，可以用任何编程语言来实现服务，只要这种编程语言可以生成服务并且可以与 WSDL 结合在一起使用就可以了。SOAP 本身并不是绝对需要的，但它是通用的消息传递系统。因此，可以使用几乎任何一种编程语言和支持 WSDL 的平台来实现 SOA 中的成员服务。

2．SOA 与 CORBA

基于 CORBA 的应用程序有许多构件必须连接到 SOA 中。虽然 CORBA 中的 IDL 在概念上类似于 WSDL，但它不是严格的，因而首先需要将其映射到 WSDL。另外，需要使用更高级的 SOA 协议（例如，用于流程和策略管理的协议），而不是 CORBA 中的类似的概念。这是 CORBA 构件（表示为服务）需要与 SOA 服务交互的情况。在 CORBA 模型中，所有的独立子集仍然可以像以前一样工作。

由 OMG 提出并在许多产品中得以实现的模型驱动的架构（Model Driven Architecture，MDA）在一个更抽象的层次上与 SOA 的概念具有很强的相关性。MDA 基于这样的概念，任何软件流程都可以定义为模型甚至是元模型（即模型的模型），然后可以将这些模型和元模型转换成应用程序的实际构件。因此，MDA 创建了一个模型，这个模型先编译成软件应用程序，而软件应用程序接着又编译成可执行程序，这样就可以在平台上运行了。MDA 并不区分服务和对象这两个概念，但是它确实允许模型由其他子集模型本身组成。

3．SOA 与自主计算

自主计算的概念应用于管理分布式服务架构的范围，具体来说，就是帮助维护策略和服务级协议以及 SOA 系统的总体稳定性。

4．SOA 与网格计算

一方面，网格计算可以以两个级别与 SOA 系统一起使用。网格是分布式计算的一种形式，它利用分布式特性和服务之间的交互来为 SOA 应用程序提供计算支持。在这种情况下，网格起到了框架的作用，其中实现了一些或所有单独的服务。因此，SOA 应用程序可以是网格服务的消费者。

另一方面，网格本身也可以构建在 SOA 之上。在这种情况下，每个操作系统服务都是构成整个 SOA 应用程序的成员，而 SOA 应用程序就是网格本身。因此，单独的网格构件既可以使用 Web 服务进行通信，又可以以 SOA 的方式进行交互。总而言之，网格系统可以是 SOA 本身，也可以提供服务来在其上构建应用程序级 SOA 模型。

有关网格的详细知识，请阅读本书第 20 章。

5．SOA 与 Web Service

在理解 SOA 和 Web Service 的关系上，经常发生混淆。根据 2003 年 4 月的 Gartner 报道，Yefim V. Natis 就这个问题是这样解释的："Web Service 是技术规范，而 SOA 是设计原则。特别是 Web Service 中的 WSDL，是一个 SOA 配套的接口定义标准。这是 Web Service 和 SOA 的根本联系" 。

从本质上来说，SOA 是一种架构模式，而 Web Service 是利用一组标准实现的服务。Web Service 是实现 SOA 的方式之一。用 Web Service 来实现 SOA 的好处是开发人员可以实现一个中立平台，来获得服务，而且随着越来越多的软件商支持越来越多的 Web Service 规范，用户会取得更好的通用性。

相对来说，Web Service 是就现在而言最适合实现 SOA 的一些技术的集合，事实上，SOA 的火爆在很大程度上归功于 Web Service 标准的成熟和应用的普及，为广泛的实现 SOA 架构提供了基础。

11.6　富互联网应用架构

互联网已经日益成为应用程序开发的默认平台，传统的 Web 应用程序是基于 HTML 页面、服务器端数据传递的模式。而 HTML 是适合于文本的，随着 Web 应用程序复杂性越来越高，传统的 Web 应用程序已经渐渐不能满足 Web 浏览者更高的、全方位的体验要求了。为此，富互联网应用（Rich Internet Application，RIA）应运而生。

11.6.1　RIA 的概念

基于 HTML 的应用程序之所以变得流行是由于应用系统的部署成本低、结构简单，且 HTML 易于学习和使用。很多用户和开发人员都乐于放弃由桌面计算机带来的用户界面改进，来实现对新数据和应用系统的快速访问。与丧失一些重要的用户界面功能相比，基于 Web 的方式所带来的好处要大得多。

然而，某些应用系统并不完全适合采用 HTML 技术。复杂的应用系统可能要求多次提取网页来完成一项事务处理，在某些领域中，如医药和财务领域，这往往导致交互速度低得无法接受。此外，虽然 HTML 开始走向简单，但是即使简单的交互活动也仍然需要用很多的脚本来完成。即使一个输入窗体经过仔细的布置和全面的脚本设计，它从浏览器所能发送的也仅仅是简单的"名字/值"对。如果一个 HTML 窗体能够以 XML 文档形式发送和接收更复杂的数据结构，那就好多了。

总的来说，传统的 Web 应用程序存在以下几个缺点：

（1）操作复杂性。由于受传统 Web 应用程序的局限性，当进行一个多步骤或多选项的事务时，用户要么会看到一份很长的、笨拙的页面；要么就得通过反复翻转若干网页、令人沮丧地执行操作步骤。

（2）数据复杂性。高效率地表达复杂的数据，是现有 Web 应用程序所面临的巨大挑战。理想的图形工具应该能够既操作简便，又能生动明了地展示各种错综复杂的数据信息。

（3）交互复杂性。互动性需求的应用程序使得交互的问题变得日益突出，用户的耐心变得越来越少，他们的要求是要向桌面应用程序的速度靠齐。

传统 Web 应用程序之所以不能够表达高度的复杂性，其主要原因是 HTML 网页技术的先天不足，当应用进行到一定深度时，这种缺陷便逐渐显露出来。一直以来，非智能的客户端提交请求并得到服务器的响应，这种以网页为载体的网络逐步形成了如今的互联网。在这种模式下，作为默认用户界面的网页，它的上下文自然地同时又是人为地进行流程分割，以便映射将业务处理分解为步骤的机制。尽管网页已经逐渐地加入越来越多的动态特点以进行改良，然而，在展示能力及与用户互动方面仍然后劲乏力。

一味地提升服务器和网络的速度既不现实又不经济，一种可行的技术方案就是采用高度互动性和局部智能型的客户端应用程序，这样，就可以在无需刷新全页或增加带宽需求的情况之下，迅速响应用户的输入并作出相应的处理。

企业级应用程序经历了几次系统架构方面的重要转变，在此过程中，客户端的表现能力有起有落。图 11-21 显示了 RIA 的发展过程。

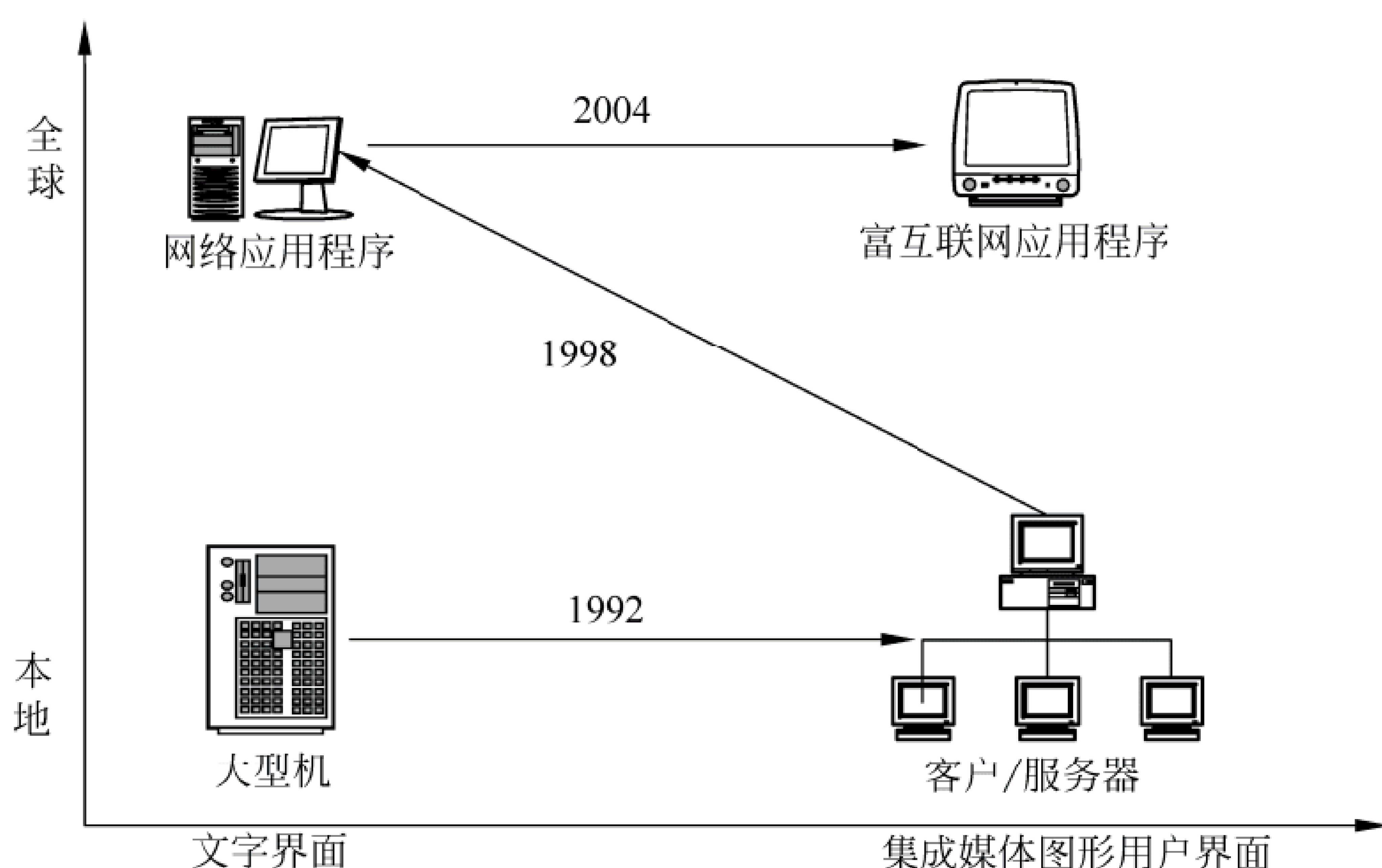


图 11-21　RIA 的发展过程

RIA 是 Web 开发和部署模式的一种演变。“富”的含义有两种，分别是丰富的数据模型和丰富的用户界面。

丰富的数据意味着客户端的用户界面能表现和应对更多更复杂的数据模式，这样才能处理客户端的运算以及异步发送、接受数据。优势在于，当页面在服务器上创建完成并交付给 HTML 后，客户端的程序为用户提供比与服务器交互更良好的感受。为了达到

高度复杂的数据模式，客户端允许用户构建一个高响应、交互式的应用程序。

丰富也指 RIA 能提供更多的改良界面。HTML 只能为用户的界面控制提供有限的功能，反之，RIA 允许一些富有创造性的界面控制，巧妙的与数据模式相合。传统的互联网模式是线性设计方式，用户唯一的选择就是用批处理方式提交页面到服务器。在这种技术限制下用户体验很糟糕，这种程序不是用户所需要的。连续处理服务器请求和页面更新存在许多障碍，包括页面响应时间、不良的网络带宽，以及满足 session 或 state 交叉连接而不断增长的日常开销。伴随着丰富的用户界面，用户可以从早期的服务器响应影响整个界面的运作模式，迁移到只对发出请求的特定区域进行改变的模式上来。本质上，意味着界面将会被分解为由单独个体组成，来适应局部改变、服务器交互，以及客户端内部构件的通信。

丰富的结果是用户可以创建一个客户端界面，这样更容易反映丰富性与复杂性共存的数据和逻辑。

RIA 利用相对健壮的客户端描述引擎，这个引擎能够提供内容密集、响应速度快和图形丰富的用户界面。除了提供一个具有各种控件（滑标、日期选择器、窗口、选项卡、微调控制器和标尺等）的界面之外，RIA 一般还允许使用可伸缩向量图（Scalable Vector Graphics，SVG）或其他技术来随时构建图形。一些 RIA 技术甚至能够提供全活动的动画来对数据变化作出响应。

RIA 的另一个好处在于，数据能够被缓存在客户端，从而可以实现一个比基于 HTML 的响应速度更快且数据往返于服务器的次数更少的用户界面。对于无线设备和需要偶尔连接的设备来说，将来的趋势肯定是向富客户端的方向发展，并且会逐渐远离基于文本的 Web 客户端。那些运行在膝上设备上的应用系统，可以被设计成以离线方式工作，或至少当连接丢失的时候能基本上以离线的方式工作。

RIA 具有的桌面应用程序的特点包括在消息确认和格式编排方面提供互动用户界面，在无刷新页面之下提供快捷的界面响应时间，提供通用的用户界面特性如拖放式（drag and drop）以及在线和离线操作能力。RIA 具有的 Web 应用程序的特点包括立即布署、跨平台、采用逐步下载来检索内容和数据以及可以充分利用被广泛采纳的互联网标准。RIA 具有通信的特点则包括实时互动的声音和图像。

客户机在 RIA 中的作用不仅是展示页面，它可以在幕后与用户请求异步地进行计算、传送和检索数据、显示集成的用户界面和综合使用声音和图像，这一切都可以在不依靠客户机连接的服务器或后端的情况下进行。

对于企业来说，部署 RIA 的好处在于：

（1）RIA 可以继续使用现有的应用程序模型（包括 J2EE 和.NET），因而无须大规模替换现有的 Web 应用程序。通过 RIA 技术，可以轻松构建更为直观、易于使用、反应更迅速并且可以脱机使用的应用程序。

（2）RIA 可以帮助企业提供多元化的重要业务效益，包括提高产品销量、提高品牌

忠诚度、延长网站逗留时间、较频繁的重复访问、减少带宽成本、减少支持求助以及增强客户关系等。

11.6.2　RIA 模型

一个在 J2EE 中典型的网络部署系统的结构如图 11-22 所示。

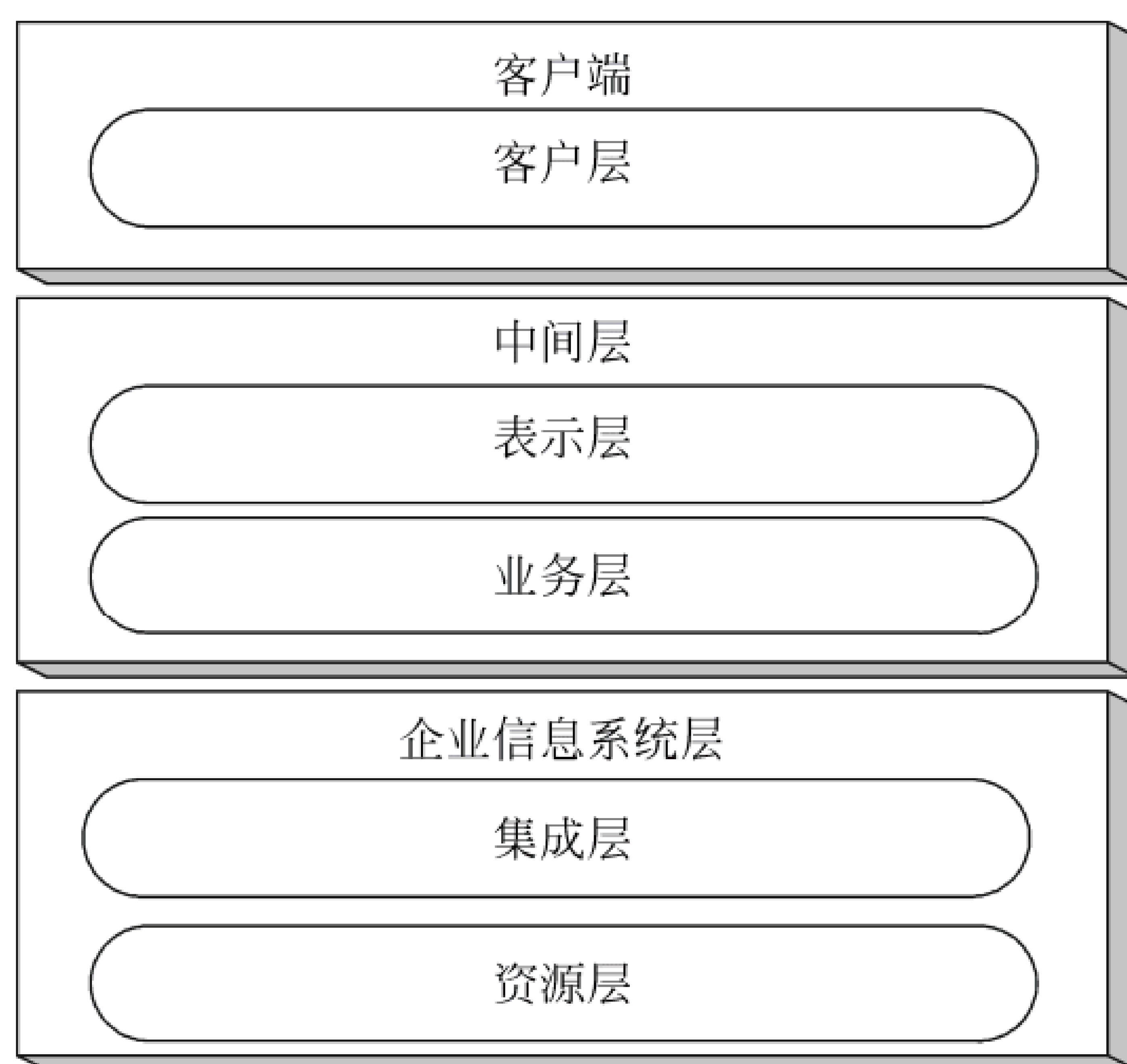


图 11-22　典型的 Web 部署系统

其中，客户层包含用户界面和程序接口的容器；表示层包含表述内容的逻辑、处理用户会话、状态管理；业务层包含系统的业务逻辑；集成层包含访问远端程序和数据源的连接器和适配器；资源层包含数据库资料以及像 ERP 这样的企业信息资源和 XML 文件。

在网络部署系统中，决定程序流程、内容创建和内容传递的逻辑是存在于中间层的，因为传统的 Web 应用程序的主要特点是一个基于客户端浏览器的请求/响应模型。如果要传递界面，系统需要在它通过中间层向客户层传递之前对内容进行以 HTML 的形式的格式化和编译。对任何表示层的修改都需要向服务器发出请求，做出响应是整个页面的而不是仅仅做出改动的部分。而在传统的网络模型中采用 Applet，Javascript/动态 HTML（Dynamic HTML，DHTML）等方法来代替客户层的解决方案，始终没有成为主流，这主要是因为各种各样的技术问题，特别是不同浏览器的兼容问题。

RIA 模型可以更好地反映出应用的结构，如图 11-23 所示。

不像 J2EE 应用那样，客户端的请求会导致系统生成一个页面再返回客户端，一个富客户可以支持更小的单元或构件，这些构件从小到一个投票问题，大到一个完整的视图或界面，富客户模型将界面分解成许多的既可以和用户直接交互，又可以和服务器进行通信的小单元模块。



图 11-23 典型的富客户端模型

这种将应用系统的设计从以一个个相对独立的页面为中心转移到以构件为中心的转变将会使客户层的设计提升到一个新的层次，并且会使客户层变得更加灵活。富客户层不再成为服务器响应的最终端，这同时也使程序的性能得以提高，用户使用的感觉就好像程序不需要和服务器进行通信或只是偶尔才需要进行通信。

RIA 模型是一种事件模型，不像传统的模型那样，服务器收到请求后由上至下的创建客户端界面，用户不用预测事件的顺序。既然每个构件都是独立的，就没有必要因为一个请求而做出影响整个视图的反应。要使每个构件都具有向服务器传送信息的能力需要每个构件知道如何处理服务器传递回来的信息。在 RIA 中，客户端和服务器端交互数据是不同步的，这样用户就可以控制构件创建信息发送给服务器和处理服务器的响应，可以为更零散的控制去耦合和分离程序功能并且建立面向服务的程序结构。

11.6.3 RIA 客户端开发技术

本节简单介绍几种常用的 RIA 客户端开发技术。

1．Macromedia Flash/Flex

Flash 是一个已经成熟的商业产品，它可以在 Web 网页中引入交互式的图形界面。最新版本包含了建立窗体风格的应用程序的功能。尽管 Flash 作为一个在 Web 上最广泛部署的前端技术还有争议，但据称已经有 98％以上的桌面系统都支持 Flash，这使得以 Macromedia Flash Player 为客户端的 RIA 可以支持种类广泛的平台和设备。由于用来创建动画式图形的 Flash 工具功能十分强大和是可视化的（与之相反其他技术要求进行低级的图形编码），所以图形设计人员使用起来十分得心应手。

Flex 是为满足希望开发 RIA 的企业级程序员的需求而推出的表示服务器和应用程序框架，它可以运行于 J2EE 和.NET 平台。Flex 表示服务器提供基于标准的、声明性的编程方法和流程，并提供运行时服务，用于开发和部署 RIA 的表示层。Flex 开发者使用直观的基于 XML 的 MXML 来定义丰富的用户界面。该语言由 Flex 服务器翻译成 SWF 格式的客户端应用程序，在 Flash Player 中运行。

Flex 和 Flash 的最大缺点在于对 XML 和 Web 服务等标准的支持很有限，而且作为应用开发工具的环境还不成熟。Flex 和 Flash 的优点在于可以很容易的用来创建复杂的动画式显示，以及可以使用第三方插件。

2．AJAX

AJAX（Asynchronous JavaScript And XML，异步 JavaScript 和 XML）用来描述一组技术，它使浏览器可以为用户提供更为自然的浏览体验。借助于 AJAX，可以在用户单击按钮时，使用 JavaScript 和 DHTML 立即更新用户界面，并向服务器发出异步请求，以执行更新或查询数据库。当请求返回时，就可以使用 JavaScript 和 CSS 来相应地更新用户界面，而不是刷新整个页面。最重要的是，用户甚至不知道浏览器正在与服务器通信，Web 站点看起来是即时响应的。AJAX 是由几种蓬勃发展的技术以新的方式组合而成的，包含基于可扩展超文本标识语言（eXtensible HyperText Markup Language，XHTML）和 CSS 标准的表示；使用 DOM 进行动态显示和交互；使用 XMLHttpRequest 与服务器进行异步通信；使用 JavaScript 绑定一切。

3．Laszlo

Laszlo 是一个开源的富客户端开发环境。使用 Laszlo 平台时，开发者只需编写名为 LZX 的描述语言（其中整合了 XML 和 JavaScript），运行在 J2EE 应用服务器上的 Laszlo 平台会将其编译成 SWF 格式的文件并传输给客户端展示。从这点上来说，Laszlo 的本质和 Flex 是一样的。Flash 是任何浏览器都支持的展示形式，从而一举解决了浏览器之间的移植问题。而且，Laszlo 还可以将 LZX 编译成 Java 或.NET 本地代码，从而大大提高运行效率。

4．Avalon

Microsoft 的 Avalon 是 Windows 的一部分，是一个图形和展示引擎，主要由新加到.NET 框架中的一组类集合而成。Avalon 定义了一个新标记语言，其代号为 XAML（可扩展应用程序标记语言）。可以使用 XAML 来定义文本、图像和控件的布局，程序代码可以直接嵌入到 XAML 中，也可以将它保留在一个单独的文件内。这与 Flex 中的 MXML 或 Laszlo 中的 LZX 非常相似。不同的是，基于 Avalon 的应用程序必须运行在 Windows 环境中，而 Flex 和 Laszlo 是不依赖于平台的，仅仅需要装有 Flash 播放器的浏览器即可。

5．Java SWT

Java 完全支持创建基于窗体的用户界面。除了 Java 基础类（JFC/Swing）中的用户界面构件之外，开发人员还可以使用来自于 Eclipse Project 的 SWT 工具箱和许多第三方

工具箱进行开发。对于图形来说，可以采用 Java 2D API，这是一个非常完整且非常复杂的图形 API。用户可以通过一个 Web 浏览器使用 Java 插件软件，或使用 Java 运行时环境中较新的 Java Web Start 技术来部署应用程序。使用 Java 建立富客户端的主要缺陷是它的复杂性（即使对简单的窗体和图形来说，也要求编写非常繁琐的代码）和 Java 浏览器插件的低市场占有率。

6. XUL

XML 用户界面语言（XML User Interface Language，XUL）来自于 Mozilla 的开放源码项目。XUL 可用于建立窗体应用程序，这些应用程序不但可以在 Mozilla 浏览器上运行，而且也可以运行在其他描述引擎上，如 Zulu（一个 Flash MX 构件）和 Thinleys（一个 Java 实现）。XUL 描述引擎都非常小（通常都在 100KB 以下），它既可以使用 XML 数据，也可以生成 XML 数据。

XUL 的一个主要缺点在于它目前还没有获得一个主要商业实体的支持。XUL 最大的优点在于它与 Gecko 引擎的集成（打开了通向大量 Web 标准的大门），以及与大多数其他 XML 用户界面描述语言相比，它是一种非常具有表达力和简洁的语言。

7. Bindows

Bindows 是用 JavaScript 和 DHTML 开发的 Web 窗体框架。JavaScript 用于客户端界面的显示和处理，XML HTTP 用于客户端与服务器的信息传输。JavaScript 在客户端的表现力不容置疑，利用 JavaScript 几乎可以实现 Windows 应用程序所能干的大部分事情，XML HTTP 一直以来常被用于实现无刷新的 Web 页面，它和 JavaScript 配合，可以完成数据从服务器和客户端的传输。

Bindows 的一个主要缺点是它采用一次全部载入的方式来实现脚本库，在窗口的加载期，需要一个漫长的等待过程，甚至浏览器的进程会产生无响应的情况。在这一点上，Bindows 根本没有遵循“用多少取多少”的准则。另外，Bindows 内部大量利用了 IE6 的技术，没有考虑到非 IE 的浏览器，限制了 Bindows 的流行。

8. Oracle Forms

Oracle Forms 是用来构建以数据库为中心的互联网应用系统的一个成熟的商品化产品。通过 Oracle Forms，用户可以使用一个输出窗体模块文件的可视化设计器创建窗体。为了便于在该设计工具外部进一步进行处理，模块文件要么采用私有的 FMT 格式，要么采用 XML 格式。这些模块文件驱动一个描述窗体的 Java 运行时环境。除了所有窗体的标准窗口小部件之外，还可以通过集成附加的可插入的 Java 构件和一些定制的 JavaBean 来实现更多的功能性。

Oracle Forms 采用的脚本语言为 PL/SQL，Oracle 数据库也采用同样的脚本语言。Oracle Forms 的一个非常有趣的特点就是，用来建立、编辑和编译窗体模块文件的 Java API——开发人员可以通过创建脚本来生成众多的窗体应用程序，或进行全局性的改动。Oracle Forms 的主要缺点是，进行 Web 部署需要获得 Oracle 应用服务器的使用许可。

Oracle Forms 的优点是，它可以与 Oracle 数据库和 Oracle 平台的其他部分紧密集成，对国际化的广泛支持，以及高效率地创建以数据为中心的应用程序。

11.7　基于架构的软件开发模型

传统的软件开发过程可以划分为从概念直到实现的若干个阶段，包括问题定义、需求分析、软件设计、软件实现及软件测试等。如果采用传统的软件开发模型，软件架构的建立应位于需求分析之后，概要设计之前。

基于架构的软件开发模型把整个基于架构的软件过程划分为架构需求、设计、文档化、复审、实现、演化等 6 个子过程，如图 11-24 所示。

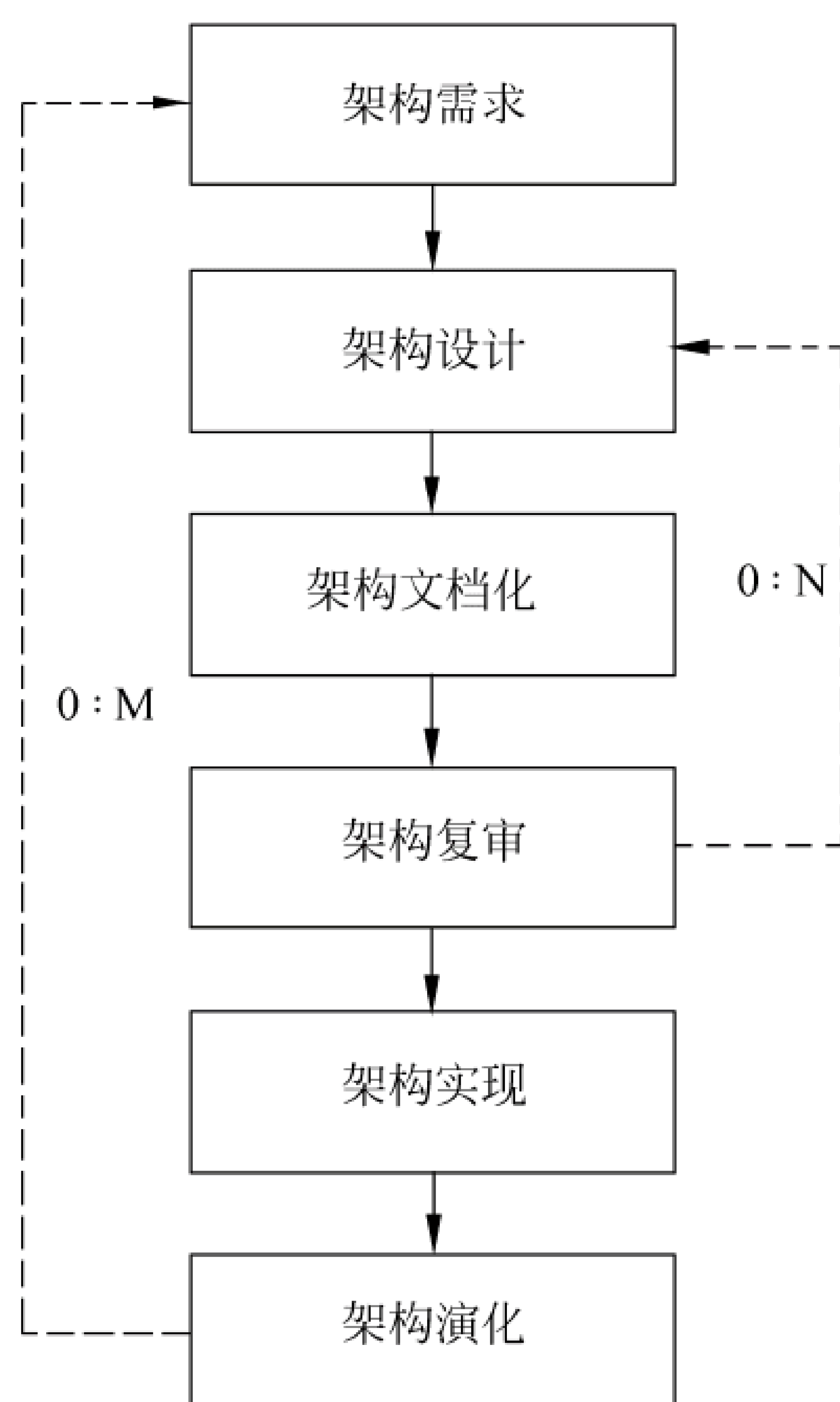


图 11-24　基于架构的软件开发模型

11.7.1　架构需求

需求是指用户对目标软件系统在功能、行为、性能、设计约束等方面的期望。架构需求受技术环境和架构设计师的经验影响。需求过程主要是获取用户需求，标识系统中所要用到的构件。架构需求过程如图 11-25 所示。如果以前有类似的系统架构的需求，可以从需求库中取出，加以利用和修改，以节省需求获取的时间，减少重复劳动，提高开发效率。

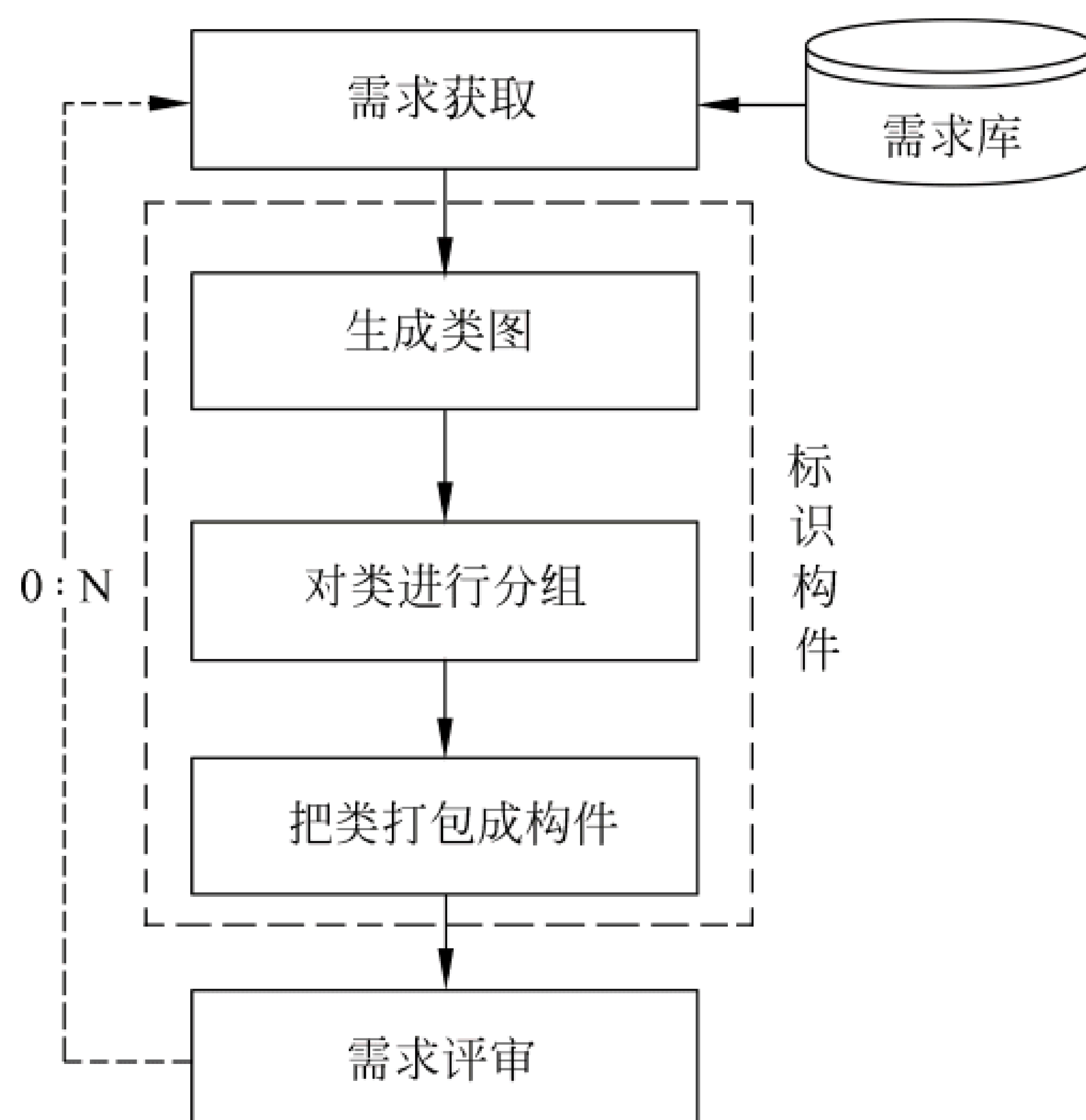


图 11-25 架构需求过程

（1）需求获取。架构需求一般来自三个方面，分别是系统的质量目标、系统的商业目标和系统开发人员的商业目标。软件架构需求获取过程主要是定义开发人员必须实现的软件功能，使得用户能完成他们的任务，从而满足业务上的功能需求。与此同时，还要获得软件质量属性，满足一些非功能需求。

（2）标识构件。在图 11-25 中虚框部分属于标识构件过程，该过程为系统生成初始逻辑结构，包含大致的构件。这一过程又可分为以下三步来实现。

第一步：生成类图。生成类图的 CASE 工具有很多，例如 Rose 就能自动生成类图。

第二步：对类进行分组。在生成的类图基础上，使用一些标准对类进行分组，可以大大简化类图结构，使之更清晰。一般地，与其他类隔离的类形成一个组，由概括关联的类组成一个附加组，由聚合或合成关联的类也形成一个附加组。

第三步：把类打包成构件。把在第二步得到的类簇打包成构件，这些构件可以分组合并成更大的构件。

（3）需求评审。组织一个由不同代表（如分析人员、客户、设计人员、测试人员）组成的小组，对架构需求及相关构件进行仔细的审查。审查的主要内容包括所获取的需求是否真实反映了用户的要求，类的分组是否合理，构件合并是否合理等。

必要时，可以在第（1）～（3）之间进行迭代。

11.7.2 架构设计

架构需求用来激发和调整设计决策，不同的视图被用来表达与质量目标有关的信息。架构设计是一个迭代过程，如果要开发的系统能够从已有的系统中导出大部分，则可以使用已有系统的设计过程。架构设计过程如图 11-26 所示。

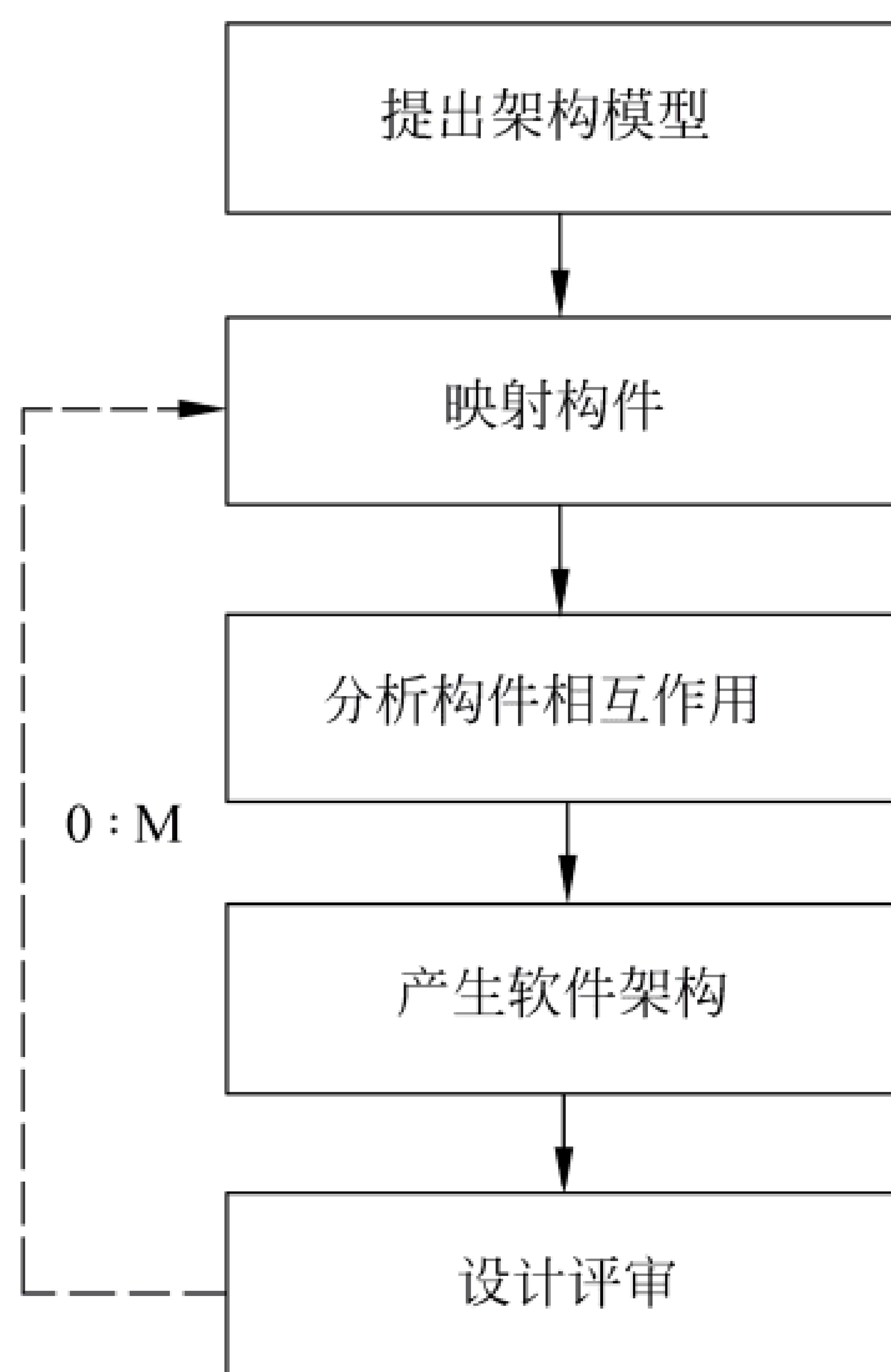


图 11-26　架构设计过程

（1）提出软件架构模型。在建立架构的初期，选择一个合适的架构风格是首要的。在这个风格基础上，开发人员通过架构模型，可以获得关于架构属性的理解。此时，虽然这个模型是理想化的（其中的某些部分可能错误地表示了应用的特征），但是，该模型为将来的实现和演化过程建立了目标。

（2）把已标识的构件映射到软件架构中。把在架构需求阶段已标识的构件映射到架构中，将产生一个中间结构，这个中间结构只包含那些能明确适合架构模型的构件。

（3）分析构件之间的相互作用。为了把所有已标识的构件集成到架构中，必须认真分析这些构件的相互作用和关系。

（4）产生软件架构。一旦决定了关键的构件之间的关系和相互作用，就可以在第 2 阶段得到的中间结构的基础上进行精化。

（5）设计评审。一旦设计了软件架构，必须邀请独立于系统开发的外部人员对架构进行评审。

11.7.3　架构文档化

绝大多数的架构都是抽象的，由一些概念上的构件组成。例如，层的概念在任何程序设计语言中都不存在。因此，要让系统分析师和程序员去实现架构，还必须得把架构进行文档化。文档是在系统演化的每一个阶段，系统设计与开发人员的通信媒介，是为验证架构设计和提炼或修改这些设计（必要时）所执行预先分析的基础。

架构文档化过程的主要输出结果是架构需求规格说明和测试架构需求的质量设计说明书这两个文档。生成需求模型构件的精确的形式化的描述，作为用户和开发者之间的一个协约。

软件架构的文档要求与软件开发项目中的其他文档是类似的。文档的完整性和质量是软件架构成功的关键因素。文档要从使用者的角度进行编写，必须分发给所有与系统有关的开发人员，且必须保证开发者手上的文档是最新的。

11.7.4 架构复审

从图 11-24 中可以看出，架构设计、文档化和复审是一个迭代过程。从这个方面来说，在一个主版本的软件架构分析之后，要安排一次由外部人员（用户代表和领域专家）参加的复审。

复审的目的是标识潜在的风险，及早发现架构设计中的缺陷和错误，包括架构能否满足需求、质量需求是否在设计中得到体现、层次是否清晰、构件的划分是否合理、文档表达是否明确、构件的设计是否满足功能与性能的要求等等。

由外部人员进行复审的目的是保证架构的设计能够公正地进行检验，使组织的管理者能够决定正式实现架构。

11.7.5 架构实现

所谓实现就是要用实体来显示出一个软件架构，即要符合架构所描述的结构性设计决策，分割成规定的构件，按规定方式互相交互。架构的实现过程如图 11-27 所示。

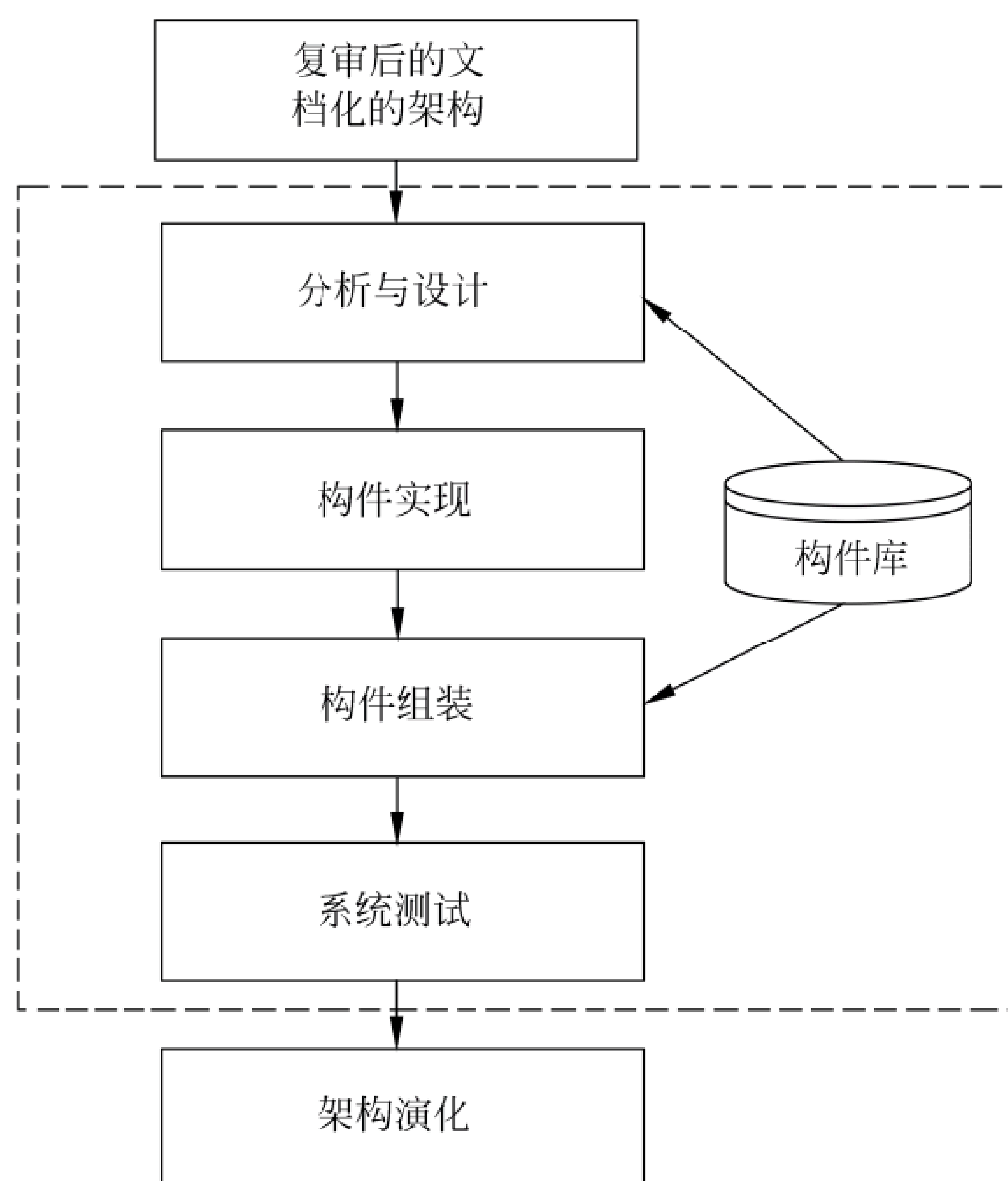


图 11-27　架构实现过程

图 11-27 中的虚框部分是架构的实现过程。整个实现过程是以复审后的文档化的架构说明书为基础的，每个构件必须满足软件架构中说明的对其他构件的责任。这些决定即实现的约束是在系统级或项目范围内作出的，每个构件上工作的实现者是看不见的。

在架构说明书中，已经定义了系统中的构件与构件之间的关系。因为在架构层次上，构件接口约束对外唯一地代表了构件，所以可以从构件库中查找符合接口约束的构件，必要时开发新的满足要求的构件。

然后，按照设计提供的结构，通过组装支持工具把这些构件的实现体组装起来，完成整个软件系统的连接与合成。

最后一步是测试，包括单个构件的功能性测试和被组装应用的整体功能和性能测试。

11.7.6　架构演化

在构件开发过程中，最终用户的需求可能还有变动。在软件开发完毕，正常运行后，由一个单位移植到另一个单位，需求也会发生变化。在这两种情况下，就必须相应地修改软件架构，以适应新的变化了的软件需求。架构演化过程如图 11-28 所示。

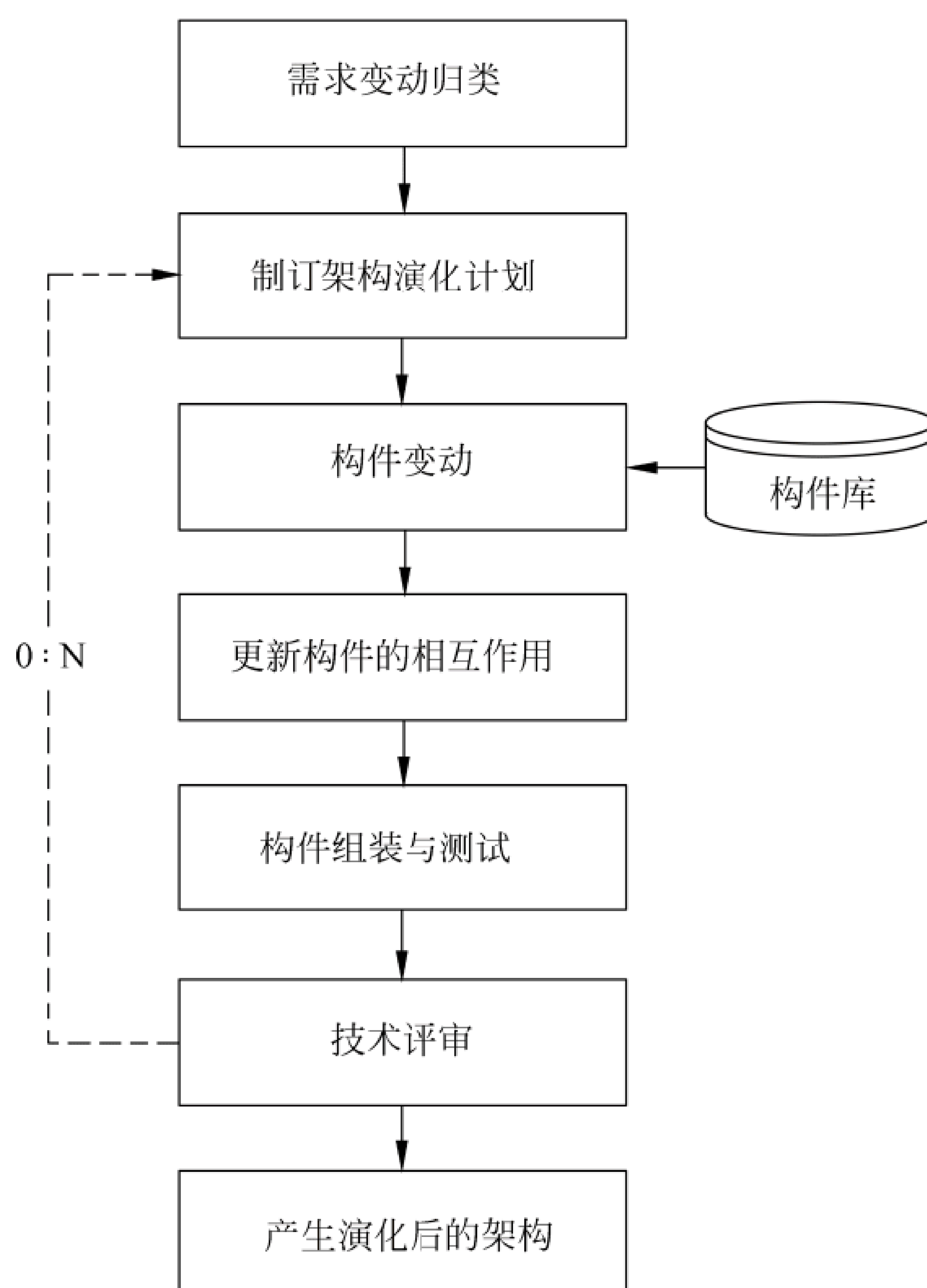


图 11-28　架构演化过程

架构演化是使用系统演化步骤去修改应用，以满足新的需求。主要包括以下 7 个步骤：

（1）需求变动归类。首先必须对用户需求的变化进行归类，使变化的需求与已有构件对应。对找不到对应构件的变动，也要作好标记，在后续工作中，将创建新的构件，以对应这部分变化的需求。

（2）制订架构演化计划。在改变原有结构之前，开发组织必须制订一个周密的架构演化计划，作为后续演化开发工作的指南。

（3）修改、增加或删除构件。在演化计划的基础上，开发人员可根据在第 1 步得到的需求变动的归类情况，决定是否修改或删除存在的构件、增加新构件。最后，对修改和增加的构件进行功能性测试。

（4）更新构件的相互作用。随着构件的增加、删除和修改，构件之间的控制流必须得到更新。

（5）构件组装与测试。通过组装支持工具把这些构件的实现体组装起来，完成整个软件系统的连接与合成，形成新的架构。然后对组装后的系统整体功能和性能进行测试。

（6）技术评审。对以上步骤进行确认，进行技术评审。评审组装后的架构是否反映需求变动，符合用户需求。如果不符合，则需要在第 2 到第 6 步之间进行迭代。

（7）产生演化后的架构。在原来系统上所作的所有修改必须集成到原来的架构中，完成一次演化过程。

11.8 软件架构评估

大家知道，软件架构的设计是整个软件开发过程中关键的一步。对于当今世界上庞大而复杂的系统来说，没有一个合适的架构而要有一个成功的软件设计几乎是不可想象的。不同类型的系统需要不同的架构，甚至一个系统的不同子系统也需要不同的架构。架构的选择往往会成为一个系统设计成败的关键。

但是，怎样才能知道为软件系统所选用的架构是恰当的呢？如何确保按照所选用的架构能顺利地开发出成功的软件产品呢？要回答这些问题并不容易，因为它受到很多因素的影响，需要专门的方法来对其进行评估。

架构评估可以只针对一个架构，也可以针对一组架构。在架构评估过程中，评估人员所关注的是系统的质量属性，包括性能、可靠性、可用性、安全性、可修改性、功能性、可变性、集成性、互操作性。

从目前已有的软件架构评估技术来看，某些技术通过与经验丰富的设计人员交流，获取他们对待评估软件架构的意见；某些技术对针对代码的质量度量进行扩展，以自底向上地推测软件架构的质量；某些技术分析把对系统的质量需求转换为一系列与系统的交互活动，分析软件架构对这一系列活动的支持程度等。尽管看起来采用的评估方式都

各不相同，但基本可以归纳为三类主要的评估方式，分别是基于调查问卷（检查表）的方式、基于场景的方式和基于度量的方式。

1．基于调查问卷或检查表的评估方式

CMU/SEI 的软件风险评估过程采用了这一方式。调查问卷是一系列可以应用到各种架构评估的相关问题，其中有些问题可能涉及到架构的设计决策；有些问题涉及到架构的文档，例如架构的表示用的是何种架构描述语言（Architecture Description Language，ADL）；有的问题针对架构描述本身的细节问题，如系统的核心功能是否与界面分开。检查表中也包含一系列比调查问卷更细节和具体的问题，它们更趋向于考察某些关心的质量属性。例如，对实时信息系统的性能进行考察时，很可能问到系统是否反复多次地将同样的数据写入磁盘等。

这一评估方式比较自由灵活，可评估多种质量属性，也可以在软件架构设计的多个阶段进行。但是，由于评估的结果很大程度上来自评估人员的主观推断，因此不同的评估人员可能会产生不同甚至截然相反的结果，而且评估人员对领域的熟悉程度、是否具有丰富的相关经验也成为评估结果是否正确的重要因素。尽管基于调查问卷与检查表的评估方式相对比较主观，但由于系统相关的人员的经验和知识是评估软件架构的重要信息来源，因而它仍然是进行软件架构评估的重要途径之一。

2．基于场景的评估方式

场景是一系列有序的使用或修改系统的步骤。基于场景的方式由 SEI 首先提出并应用在架构权衡分析方法（Architecture Tradeoff Analysis Method，ATAM）和软件架构分析方法（Software Architecture Analysis Method，SAAM）中。这种软件架构评估方式分析软件架构对场景也就是对系统的使用或修改活动的支持程度，从而判断该架构对这一场景所代表的质量需求的满足程度。例如，用一系列对软件的修改来反映易修改性方面的需求，用一系列攻击性操作来代表安全性方面的需求等。

这一评估方式考虑到了包括系统的开发人员、维护人员、最终用户、管理人员、测试人员等在内的所有与系统相关的人员对质量的要求。基于场景的评估方式涉及到的基本活动包括确定应用领域的功能和软件架构的结构之间的映射，设计用于体现待评估质量属性的场景以及分析软件架构对场景的支持程度。

不同的应用系统对同一质量属性的理解可能不同，例如，对操作系统来说，可移植性被理解为系统可在不同的硬件平台上运行，而对于普通的应用系统而言，可移植性往往是指该系统可在不同的操作系统上运行。由于存在这种不一致性，对一个领域适合的场景设计在另一个领域内未必合适，因此基于场景的评估方式是特定于领域的。这一评估方式的实施者一方面需要有丰富的领域知识，以对某一质量需求设计出合理的场景；另一方面，必须对待评估的软件架构有一定的了解，以准确判断它是否支持场景描述的一系列活动。

3．基于度量的评估方式

度量是指为软件产品的某一属性所赋予的数值，例如，代码行数、方法调用层数、构件个数等。传统的度量研究主要针对代码，但近年来也出现了一些针对高层设计的度量，软件架构度量即是其中之一。代码度量和代码质量之间存在着重要的联系，类似地，软件架构度量应该也能够作为评判质量的重要的依据。赫尔辛基大学提出的基于模式挖掘的面向对象软件架构度量技术、Karlskrona 和 Ronneby 提出的基于面向对象度量的软件架构可维护性评估、西弗吉尼亚大学提出的软件架构度量方法等都在这方面进行了探索，提出了一些可操作的具体方案。把这类评估方式称作基于度量的评估方式。

上述基于度量的评估技术都涉及三个基本活动：首先，需要建立质量属性和度量之间的映射原则，即确定怎样从度量结果推出系统具有什么样的质量属性；然后，从软件架构文档中获取度量信息；最后，根据映射原则分析、推导出系统的某些质量属性。因此，这些评估技术被认为都采用了基于度量的评估方式。

基于度量的评估方式提供更为客观和量化的质量评估。这一评估方式需要在软件架构的设计基本完成以后才能进行，而且需要评估人员对待评估的架构十分了解。否则，不能获取准确的度量。自动的软件架构度量获取工具能在一定程度上简化了评估的难度，例如，MAISA 可从文本格式的 UML 图中抽取面向对象架构的度量。

4．比较

经过对三类主要的软件架构质量评估方式的分析，用表 11-1 从通用性、评估者对架构的了解程度、评估实施阶段、评估方式的客观程度、工具支持程度等方面对这三种方式进行简单的比较。

表 11-1　三类评估方式比较表

评 估 方 式	调查问卷或检查表		场　　景	度　　量
	调查问卷	检查表		
通用性	通用	特定领域	特定系统	通用或特定领域
评估者对架构的了解程度	粗略了解	无限制	中等了解	精确了解
实施阶段	早	中	中	中
客观性	主观	主观	较主观	较客观

本章参考文献

[1] 张友生. 软件体系结构（第二版）. 北京：清华大学出版社，2006

[2] 孙昌爱，金茂忠，刘超. 软件体系结构研究综述. 软件学报，2002（7）：1228-1237

[3] 叶俊民，赵恒，曹瀚等. 软件体系结构风格的实例研究. 小型微型计算机系统，2003（10）：1158-1160

[4] 李克勤，陈兆良，梅宏等. 领域工程概述. 计算机科学，1999

[5] 周欣，黄璜. 软件体系结构质量评价概述. 计算机科学，2003（1）：49-52
[6] 黄启春. 拨开 SOA 的面纱. http://tech.csai.cn/sa/no000079.htm
[7] 崔晓波. SOA 概览. http://51cmm.csai.cn/Monograph/200611141116331889.htm
[8] 仲萃豪. SOA 的十大技术理论体系. http://51cmm.csai.cn/Monograph/
[9] 庞引明. SOA 框架的不足. http://51cmm.csai.cn/Monograph/
[10] IBM DW. SOA 的生命周期. http://51cmm.csai.cn/Monograph/
[11] 中国 RIA 开发者论坛. 迎接 RIA 时代的来临. http://develop.csai.cn/web/
[12] CSAI 网. RIA 目前的发展态势及未来预测. http://develop.csai.cn/web/
[13] 中国 RIA 开发者论坛. RIA 研究与开发. http://develop.csai.cn/web/

第 12 章　设 计 模 式

随着面向对象技术的出现和广泛使用，一方面软件的可重用性在一定程度上已经有所解决，另一方面对软件可重用性的要求同时也越来越高。设计面向对象的软件很难，而设计可重复使用的面向对象的软件难度更大。开发人员必须找到适当的对象，将它们分解到粒度合适的类、定义类接口和继承体系，并建立它们之间的关键联系。

在某些时候，设计师的设计可能是针对当前的具体问题而进行的，但它应该可能通用到足以适应未来的问题和需求。因为他们总是希望避免重复设计，至少将之减少到最低水平。在一个设计完成之前，有经验的面向对象的设计师往往要重复使用若干次，而且每次都要进行改进。他们知道，不能只用最初的方法解决每个问题，常常重复使用那些过去用过的解决方案。当他们找到一个好的解决方案时，总是一次又一次地使用它。这些经验也正是他们成为专家的法宝，这就是设计经验的价值。

因此，可将设计面向对象软件的经验记录成“设计模式”（Design Pattern）。每个设计模式都有系统的命名、解释和评价了面向对象系统中一个重要的设计。目标是将设计经验收集成人们可以有效利用的模型。为此，可以记录一些最重要的设计模式，并以目录形式表现出来。

利用设计模式可方便地重用成功的设计和结构。把已经证实的技术表示为设计模式，使它们更加容易被新系统的开发者所接受。设计模式帮助设计师选择可使系统重用的设计方案，避免选择危害到可重用性的方案。设计模式还提供了类和对象接口的明确的说明书和这些接口的潜在意义，来改进现有系统的记录和维护。

12.1　设计模式概述

设计模式的概念最早是由一位叫做 christopher Alexander 的建筑学家提出来的，他试图找到一种结构化、可重用的方法，以在图纸上捕捉到建筑物的基本要素。他把注意力放在建筑物和城镇的设计和结构上，可是逐渐地他的思想影响了软件研究，并逐渐流行起来。Alexander 提出的模式是指经过时间考验的解决方案，使用模式可以降低解决问题的复杂度。在编程时，很多情况下代码都不是从头编写，而是经过模仿得来，即从别处搬过来，再经过一定改造使之适应当前情况。设计模式可以视为这种模仿的一种抽象，包含一组规则，描述了如何在软件开发领域中完成一定的任务。从这个意义上讲，所有的算法都属于编程领域的设计模式。

在介绍设计模式的具体定义之前，先看一个例子：模型-视图-控制器（Model-View-

Controler，MVC），在开发人机界面软件时考虑这种模式。用户界面承担着向用户显示问题模型、与用户进行操作、输入/输出交互的作用。用户希望保持交互操作界面的相对稳定，但更希望根据需要改变和调整显示的内容和形式。例如，要求支持不同的界面标准或得到不同的显示效果，适应不同的操作需求。这就要求界面结构能够在不改变软件功能的情况下，支持用户对界面结构的调整。要做到这一点，从界面构成的角度看，困难在于：在满足对界面要求的同时，如何使软件的计算模型独立于界面的构成。MVC 就是这样的一种交互界面的结构组织模型。

对于界面设计可变性的需求，MVC 把交互系统的组成分解成模型、视图、控制三种构件。其中模型构件独立于外在显示内容和形式，是软件所处理的问题逻辑的内在抽象，它封装了问题的核心数据、逻辑和功能的计算关系，独立于具体的界面表达和输入/输出操作；视图构件把表示模型数据及逻辑关系和状态的信息以特定形式展示给用户，它从模型获得显示信息，对于相同的信息可以有多个不同的显示形式或视图；控制构件处理用户与软件的交互操作，其职责是决定软件的控制流程，确保用户界面与模型间的对应联系，它接受用户的输入，将输入反馈给模型，进而实现对模型的计算控制，它是使模型和视图协调工作的部件。

模型、视图与控制器的分离，使得一个模型可以具有多个显示视图。如果用户通过某个视图的控制器改变了模型的数据，所有其他依赖于这些数据的视图都应反映出这些变化。因此，无论何时发生了何种数据变化，控制器都会将变化通知所有的视图，导致显示的更新。图 12-1 描述了 MVC 解决方案。

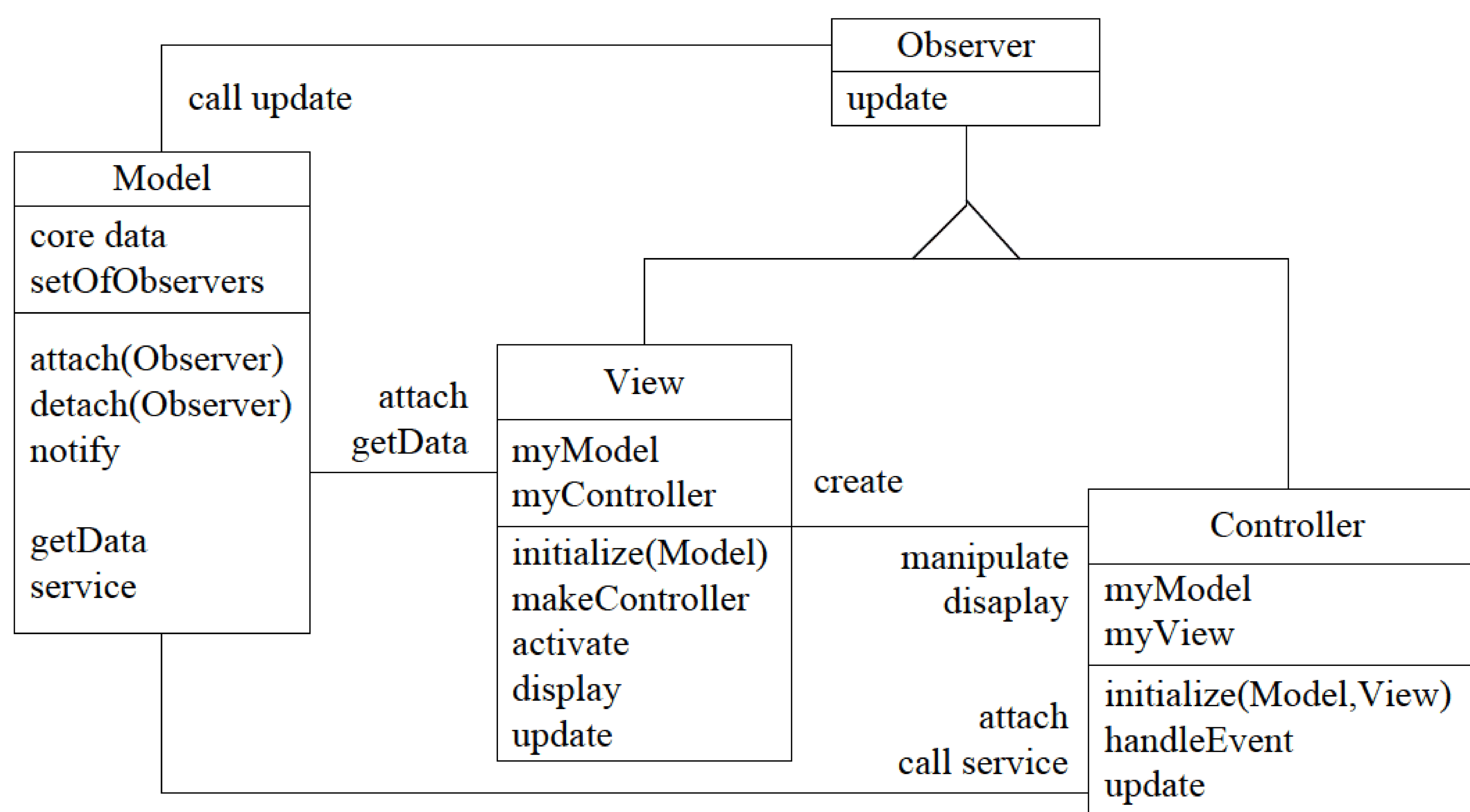


图 12-1　MVC 解决方案

从上面的例子中，可以导出设计模式的下列属性：一个模式关注一个在特定设计环境中出现的重现设计问题，并为它提供一个解决方案。在上面的例子中，问题是支持用户界面的可变性。开发人机交互软件系统时，这个问题就会出现。

所谓设计模式，简单地理解，是一些设计面向对象的软件开发的经验总结。一个设计模式事实上是系统地命名、解释和评价某一个重要的可重现的面向对象的设计方案。正如 Alexander 所说的："每一个模式描述了一个在我们身边一再发生的问题，它告诉我们这个问题的解的关键，以使你可以成千上万次地利用这个解，而不需要再一次去解它"。尽管 Alexander 所说的是有关建筑和城镇的模式，它同样适用于面向对象的设计模式，只不过要解决的问题是软件开发中一再出现的问题。

模式的概念是随设计中要解决的问题的变化而变化的。更明确地说，重复发生的具体形式就是这一重复出现的问题的解。但是一个模式又并不仅仅是它的解。问题是在一个特殊的环境中发生的，因此有很多复杂的考虑因素。给定一个环境，所提出的问题包含了一些平衡各方面考虑的结果，或称为权衡。使用模式的形式，解决方案的描述可以把握住方案所体现的本质，故而别人可以从中学到一些东西，进而在相似的情况下可以进行应用。每个模式都有一个名字，帮助开发人员讨论模式和它所给出的信息。

希赛教育专家提示：在工作中要不断总结、不断回顾以前使用过的设计模式。不同的模式之间有联系，同时也有各自的优缺点，在应用中要注意仔细考虑、权衡利弊、加以取舍。只有这样，才可能真正用好设计模式。

12.2 设计模式的组成

一个好的模式必须做到以下几点。

（1）解决一个问题：从模式可以得到解，而不仅仅是抽象的原则或策略。

（2）是一个被证明了的概念：模式通过一个记录得到解，而不是通过理论或推测。

（3）解并不是显然的：许多解决问题的方法（例如，软件设计范例或方法）是从最基本的原理得到解，而最好的方式是以非直接的方式得到解，对大多数比较困难的设计问题来说，这是必要的。

（4）描述了一种关系：模式并不仅仅描述模块，它给出更深层的系统结构和机理。

（5）模式有重要的人为因素：所有的软件服务于人类的需求，而最好的模式追求它的实用性和美学。

12.2.1 设计模式的基本成分

一般来说，一个模式由 4 个基本成分组成，分别是模式名称、问题、解决方案、效果。

1．模式名称

模式名称通常用来描述一个设计问题、它的解法和效果，由一到两个词组成。模式名称的产生使开发人员可以在更高的抽象层次上进行设计并交流设计思想。因此，寻找好的模式名称是一个很重要也是很困难的工作。

2．问题

问题告诉我们，什么时候要使用设计模式、解释问题及其背景。例如，MVC 模式关心用户界面经常变化的问题。它可能描述诸如如何将一个算法表示成一个对象这样的特殊设计问题。在应用这个模式之前，也许还要给出一些该模式的适用条件。

模式的问题陈述用一个强制条件(force)集来表示，该词最初是从建筑学和 Alexander 那里借用来的，模式组织使用术语"强制条件"来说明问题要解决时应该考虑的各个方面，例如：

（1）解决方案必须满足的需求。例如，对等进程间通信必须是高效的。

（2）必须考虑的约束。例如，进程间通信必须遵循特定协议。

（3）解决方案必须具有期望的特性。例如，软件更改应该是容易的。

MVC 模式指出了两个强制条件：它必须易于修改用户界面，但软件的功能核心不能被修改所影响。一般地，强制条件从多个角度讨论问题并有助于设计师了解它的细节。强制条件可以相互补充或相互矛盾。例如，系统的可扩展性与代码的最小化构成了两个相互矛盾的强制条件。如果希望系统可扩展，那么就应倾向于使用抽象超类；如果想使代码最小化（如用于嵌入式系统），就不能承受抽象超类的奢侈。但更重要的是，强制条件是解决问题的关键。它们平衡得越好，对问题的解决方案就越好。所以，强制条件的详细讨论是问题陈述的重要部分。

3．解决方案

解决方案描述设计的基本要素，它们的关系、各自的任务以及相互之间的合作。解决方案并不是针对某一个特殊问题而给出的。设计模式提供有关设计问题的一个抽象描述以及如何安排这些基本要素以解决问题。一个模式就像一个可以在许多不同环境下使用的模板，抽象的描述使我们可以把该模式应用于解决许多不同的问题。

模式的解决方案部分给出了如何解决再现问题，或更恰当地说是如何平衡与之相关的强制条件。在软件架构中，这样的解决方案包括两个方面。

（1）每个模式规定了一个特定的结构，即元素的一个空间配置。例如，MVC 模式的描述包括以下语句："把一个交互应用程序划分成三部分：处理、输入和输出"。

（2）每个模式规定了运行期间的行为。例如，MVC 模式的解决方案部分包括以下陈述："控制器接收输入，而输入往往是鼠标移动、点击鼠标按键或键盘输入等事件。事件转换成服务请求，这些请求再发送给模型或视图"。

希赛教育专家提示：解决方案不必解决与问题相关的所有强制条件，可以集中于特殊的强制条件，而对于剩下的强制条件进行部分解决或完全不解决，特别是强制条件相

互矛盾时。

4．效果

效果描述应用设计模式后的结果和权衡。比较与其他设计方法的异同，得到应用设计模式的代价和优点。对于软件设计来说，通常要考虑的是空间和时间的权衡，也会涉及语言问题和实现问题。对于一个面向对象的设计而言，可重用性很重要，效果还包括对系统灵活性、可扩充性及可移植性的影响，明确看出这些效果有助于理解和评价设计模式。

另外，不同的观点会影响人们对什么是设计模式的解释。某一个人的模式对另一个人来说可能只是一个基本的构造块。这里把设计模式处理到一定的抽象程度，它不用于直接编码或类重用，也不是复杂到可作为一个完整的应用或子系统的领域专用的设计，而是对一定的对象与类的关系进行描述，进而可对其进行一定程度的修改，使之可解决在一定条件下的通用设计问题。

设计模式命名、抽象并确定了一个普遍的设计结构的关键方面。这些方面有助于得到可重用的面向对象的设计。设计模式确定了参与的类和实例、它们的地位和协作，以及责任的分配。每一个设计模式都集中于特定的面向对象设计问题，描述了何时使用、是否能在其他设计约束条件下使用及使用后的效果。

12.2.2 设计模式的描述

如果要理解和讨论模式，就必须以适当形式描述模式。好的描述有助于设计师立即抓住模式的本质，即模式关心的问题是什么，以及提出的解决方案是什么。

模式也应该以统一的方式来描述。这有助于对模式进行比较，尤其在为一个问题寻求可选择的解决方案时。那么，如何描述一个设计模式呢？仅仅依靠图示的方法是不够的。尽管图示的方法很重要也很有用，但它们只能把设计的最终结果表示成一些类和对象的关系。事实上，为了重用该设计，还应该记录下产生这个设计的决策和权衡过程。具体的实例也很重要，从中可以看到设计模式的运转过程。Alexander 采用下面的格式来描述设计模式：

```
IF    you find yourself in CONTEXT
       For example EXAMPLES,
       With PROBLEM,
       Entailing FORCESS
THEN   for some REASONS,
       Apply DESIGN FORM AND/OR RULE
       To construct SOLUTION
      Leading to NEW CONTEXT and OTHER PATTERNS
```

Erich Gamma 等人采用下面的固定模式来描述，这也是目前最常用的格式。

（1）模式名称和分类：模式名称和一个简短的摘要。

（2）目的：回答下面的问题，即本设计模式的用处、它的基本原理和目的、它针对的是什么特殊的设计问题。

（3）别名：由于设计模式的提取是由许多专家得到的，同一个模式可能会被不同的专家冠以不同的命名。

（4）动机：描述一个设计问题的方案，以及模式中类和对象的结构是如何解决这个问题的。

（5）应用：在什么情况下可以应用本设计模式，如何辨认这些情况。

（6）结构：用对象模型技术对本模式的图像表示。另外，也给出了对象间相互的要求和合作的内在交互图。

（7）成分：组成本设计模式的类和对象及它们的职责。

（8）合作：成分间如何合作实现它们的任务。

（9）效果：该模式如何支持它的对象；如何在使用本模式时进行权衡，即其结果如何；可以独立地改变系统结构的哪些方面。

（10）实现：在实现本模式的过程中，要注意哪些缺陷、线索或技术；是否与编程语言有关。

（11）例程代码：说明如何用 C++或其他语言来实现该模式的代码段。

（12）已知的应用：现实系统中使用该模式的实例。

（13）相关模式：与本模式相关的一些其他模式，它们之间的区别，以及本模式是否要和其他模式共同使用。

在特定的软件开发领域中，可以用不同的描述方法。上面的 13 个要素可以忽略或合并。

12.3　设计模式的分类

对于设计模式的分类，从不同的角度则有不同的分类方法。本节介绍几种常见的分类方法，在这些方法中，主要掌握的是 GoF 模式。

1．Coad 的面向对象模式

1992 年，Coad 从 MVC 的角度对面向对象系统进行了讨论，设计模式由最底层的构成部分（类和对象）及其关系来区分。他使用了一种通用的方式来描述一种设计模式：

（1）模式所能解决的问题的简要介绍与讨论。

（2）模式的非形式文本描述以及图形表示。

（3）模式的使用方针：在何时使用，以及能够与哪些模式结合使用。

可以将 Coad 的模式划分为以下三类。

（1）基本的继承和交互模式：主要包括面向对象程序设计语言所提供的基本建模功

能。继承模式声明了一个类能够在其子类中被修改或被补充，交互模式描述了在有多个类的情况下消息的传递。

（2）面向对象软件系统的结构化模式：描述了在适当情况下，一组类如何支持面向对象软件系统结构的建模。主要包括条目（item）描述模式、为角色变动服务的设计模式和处理对象集合的模式。

（3）与 MVC 框架相关的模式。

几乎所有 Coad 提出的模式都指明如何构造面向对象的软件系统，有助于设计单个的或一小组构件，描述了 MVC 框架的各个方面。但是，他没有重视抽象类和框架，没有说明如何改造框架。

2．代码模式

代码（coding）模式的抽象方式与面向对象程序设计语言中的代码规范很相似，该类模式有助于解决某种面向对象程序设计语言中的特定问题。代码模式的主要目标在于：

（1）指明结合基本语言概念的可用方式。

（2）构成源码结构与命名规范的基础。

（3）避免面向对象程序设计语言（尤其是 C++语言）的缺陷。

代码模式与具体的程序设计语言或类库有关，它们主要从语法的角度对于软件系统的结构方面提供一些基本的规范。这些模式对于类的设计不适用，同时也不支持程序员开发和应用框架，命名规范是类库中的名字标准化的基本方法，以免在使用类库时产生混淆。

3．框架应用模式

在应用程序框架菜谱（Application Framework Cookbook Recipes）中有很多“菜谱条”，它们用一种不很规范的方式描述了如何应用框架来解决特定的问题。程序员将框架作为应用程序开发的基础，特定的框架适用于特定的需求。菜谱条通常并不包括框架的内部设计实现，只包括如何使用。

实践证明，菜谱的概念非常适合于框架的应用，它覆盖了大部分典型的框架应用，但是这些菜谱基本上都是不完全的。在菜谱中说明的应用情况越多，就越不容易找到相应的菜谱条，并且有的应用可以用数种方案来解决，或要用数种方案的结合来解决，这种交错结构的不清晰性使程序员很容易糊涂。为了避免这样的问题，菜谱应该由那些对框架本身有相当深入的理解的人来撰写，最理想的情况是由框架的开发者来撰写。

超文本系统能够很好地支持这种菜谱方法，更高级的超文本系统（如 ET++）已经超出了简单的应用菜谱的范畴，它们还可以基于设计模式方法（如设计模式目录、元模式等）来对框架的设计做文档。

4．形式合约

形式合约（Formal Contracts）也是一种描述框架设计的方法，强调组成框架的对象间的交互关系。有人认为它是面向交互的设计，对其他方法的发展有启迪作用。但形式

化方法由于其过于抽象，而有很大的局限性，仅仅在小规模程序中使用。

Helm 等人是形式合约模式的倡导者，他们最先在面向对象系统领域内探索用抽象的方法来描述被他们称为行为合成（Behavioral Composition）的内容。他们所使用的规范符号有如下优点：

（1）符号所包含的元素很少，并且其中引入的概念能够被映射成为面向对象程序设计语言中的概念。例如，参与者映射成为对象。

（2）形式合约中考虑到了复杂行为是由简单行为组成的事实，合约的修订和扩充操作使得这种方法很灵活，易于应用。

形式合约模式的缺点有以下三点：

（1）在某些情况下很难用，过于繁琐。若引入新的符号，则又使符号系统复杂化。

（2）强制性地要求过分精密，从而在说明中可能发生隐患（例如冗余）。

（3）形式合约的抽象程度过低，接近面向对象的程序设计语言，不易分清主次。

5．设计模式目录的内容

Erich Gamma 在他的博士论文中总结了一系列的设计模式，做出了开创性的工作。他用一种类似分类目录的形式将设计模式记载下来。称这些设计模式为设计模式目录。根据模式的目标（所做的事情），可以将它们分成创建性模式（creational）、结构性模式（structural）和行为性模式（behavioral）。创建性模式处理的是对象的创建过程，结构性模式处理的是对象/类的组合，行为性模式处理类和对象间的交互方式和任务分布。根据它们主要的应用对象，又可以分为主要应用于类的和主要应用于对象的。

表 12-1 是 Erich Gamma 等人总结的 23 种设计模式，这些设计模式通常被称为四人组（Gang of Four，GoF）模式。因为这些模式是在 *Design Patterns*：*Elements of Reusable Object-Oriented Software* 中正式提出的，而该书的作者是 Erich Gamma、Richard Helm、Ralph Johnson 和 John Vlissides，这几位作者常被称为“四人组”。

表 12-1　设计模式目录的分类

目　的	设 计 模 式	简 要 说 明	可改变的方面
创建型	Abstract Factory 抽象工厂模式	提供一个接口，可以创建一系列相关或相互依赖的对象，而无需指定它们具体的类	产品对象族
	Builder 生成器模式	将一个复杂类的表示与其构造相分离，使得相同的构建过程能够得出不同的表示	如何建立一种组合对象
	Factory Method* 工厂方法模式	定义一个创建对象的接口，但由子类决定需要实例化哪一个类。工厂方法使得子类实例化的过程推迟	实例化子类的对象
	Prototype 原型模式	用原型实例指定创建对象的类型，并且通过复制这个原型来创建新的对象	实例化类的对象

续表

目　的	设 计 模 式	简 要 说 明	可改变的方面
创建型	Singleton 单子模式	保证一个类只有一个实例，并提供一个访问它的全局访问点	类的单个实例
结构型	Adapter* 适配器模式	将一个类的接口转换成用户希望得到的另一种接口。它使原本不相容的接口得以协同工作	与对象的接口
	Bridge 桥模式	将类的抽象部分和它的实现部分分离开来，使它们可以独立地变化	对象的实现
	Composite 组合模式	将对象组合成树型结构以表示“整体-部分”的层次结构，使得用户对单个对象和组合对象的使用具有一致性	对象的结构和组合
	Decorator 装饰模式	动态地给一个对象添加一些额外的职责。它提供了用子类扩展功能的一个灵活的替代，比派生一个子类更加灵活	无子类对象的责任
	Façade 外观模式	定义一个高层接口，为子系统中的一组接口提供一个一致的外观，从而简化了该子系统的使用	与子系统的接口
	Flyweight 享元模式	提供支持大量细粒度对象共享的有效方法	对象的存储代价
	Proxy 代理模式	为其他对象提供一种代理以控制这个对象的访问	如何访问对象，对象位置
行为型	Chain of Responsibility 职责链模式	通过给多个对象处理请求的机会，减少请求的发送者与接收者之间的耦合。将接收对象链接起来，在链中传递请求，直到有一个对象处理这个请求	可满足请求的对象
	Command 命令模式	将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化，将请求排队或记录请求日志，支持可撤销的操作	何时及如何满足一个请求
	Interpreter* 解释器模式	给定一种语言，定义它的文法表示，并定义一个解释器，该解释器用来根据文法表示来解释语言中的句子	语言的语法和解释
	Iterator 迭代器模式	提供一种方法来顺序访问一个聚合对象中的各个元素，而不需要暴露该对象的内部表示	如何访问、遍历聚合的元素
	Mediator 中介者模式	用一个中介对象来封装一系列的对象交互。它使各对象不需要显式地相互调用，从而达到低耦合，还可以独立地改变对象间的交互	对象之间如何交互及哪些对象交互
	Memento 备忘录模式	在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，从而可以在以后将该对象恢复到原先保存的状态	何时及哪些私有信息存储在对象之外

续表

目　的	设 计 模 式	简 要 说 明	可改变的方面
行为型	Observer 观察者模式	定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动更新	依赖于另一对象的数量
	State 状态模式	允许一个对象在其内部状态改变时改变它的行为	对象的状态
	Strategy 策略模式	定义一系列算法，把它们一个个封装起来，并且使它们之间可互相替换，从而让算法可以独立于使用它的用户而变化	算法
	Template Method* 模板模式	定义一个操作中的算法骨架，而将一些步骤延迟到子类中，使得子类可以不改变一个算法的结构即可重新定义算法的某些特定步骤	算法的步骤
	Visitor 访问者模式	表示一个作用于某对象结构中的各元素的操作，使得在不改变各元素的类的前提下定义作用于这些元素的新操作	无需改变其类而可应用于对象的操作

其中带*为关于类的，其他是关于对象的。

12.4　设计模式的实现

本节以 Abstract Factory 模式为例，介绍设计模式的具体实现。其他设计模式的实现，请参考有关设计模式方面的专门书籍。

1．模式名称

Abstract Factory，也经常称之为抽象工厂模式。

2．意图解决的问题

在程序中创建一个对象似乎是不能再简单的事情，其实不然。在大型系统开发中：

（1）object　new ClassName；是最常见的创建对象的方法，但这种方法造成类名的硬编码，需要根据不同的运行环境动态加载接口相同但实现不同的类实例，这样的创建方法就需要配合复杂的判断，实例化为不同的对象。

（2）为了适用于不同的运行环境，经常使用抽象类定义接口，并在不同的运行环境中实现这个抽象类的子类。普通的创建方式必然造成代码与运行环境的强绑定，软件产品无法移植到其他的运行环境。

抽象工厂模式就可以解决这样的问题，根据不同的配置或上下文环境加载具有相同接口的不同类实例。

3．模式描述

Abstract Factory 模式的结构如图 12-2 所示。

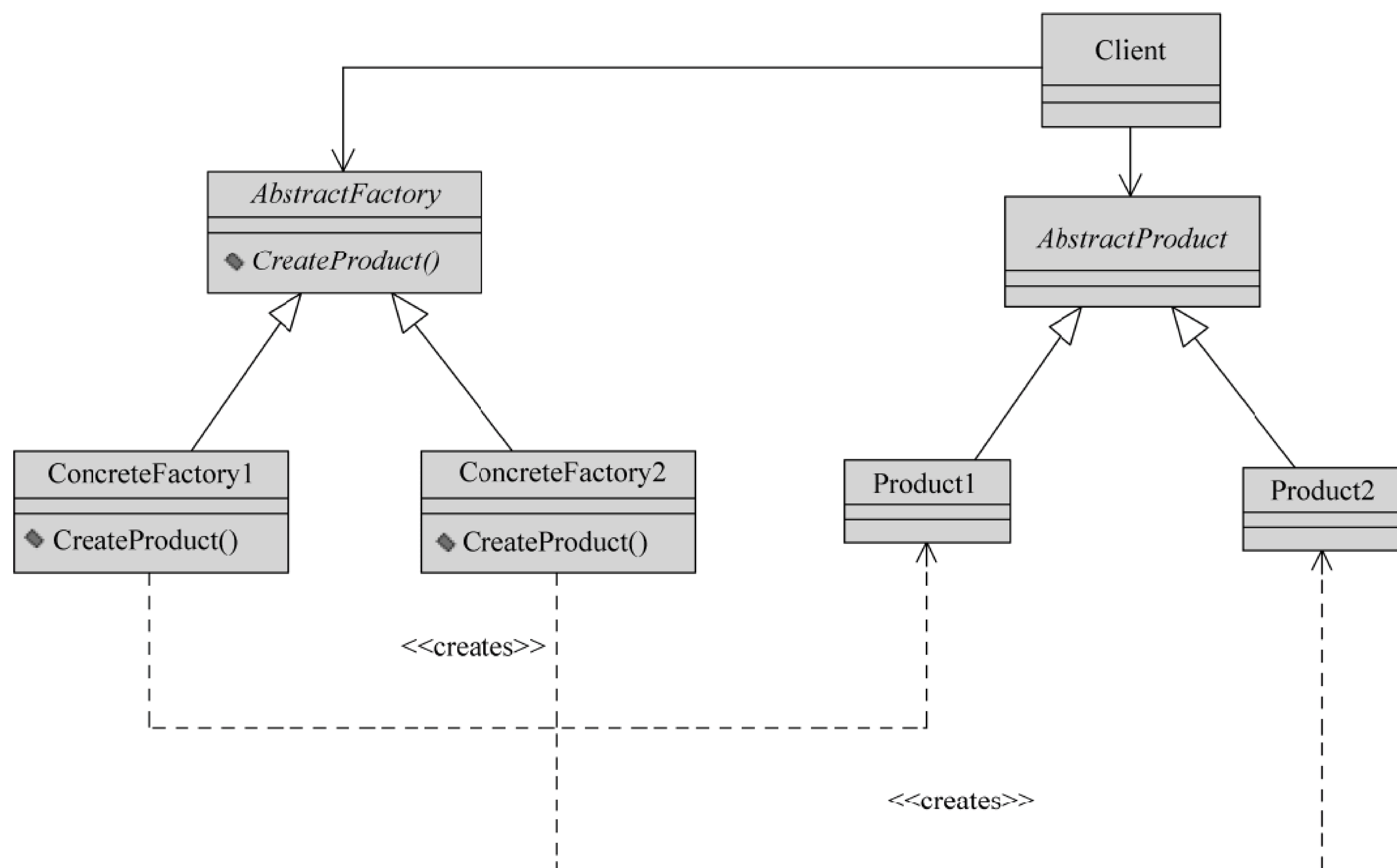


图 12-2 Abstract Factory 结构图

就如同抽象工厂的名字一样，Abstract Factory 类将接受 Client 的“订单”（Client 发送过来的消息），使用不同的“车间”（不同的 Concrete Factory），根据已有的“产品模型”（Abstract Product），生产出特定的“产品”（Product）。不同的车间生产出不同的产品供客户使用，车间与产品的关系是一一对应的。由于所有的产品都遵循产品模型，具有相同的接口，所以，这些产品都可以直接交付客户使用。

在抽象工厂模式中，AbstractFactory 可以有多个类似于 Create Product()的虚方法，就如同一个工厂中有多条产品线一样。Create Product1()创建产品线 1，Create Product2()创建产品线 2。需要注意的是，一般在 Abstract Factory 中的 Create Product()方法与 Abstract Product 是一一对应的。而 Concrete Factory 的数量与实际的 Product 的数量是一致的。

4. 效果

应用 Abstract Factory 模式可以实现对象可配置的、动态的创建。灵活运用 Abstract Factory 模式可以提高软件产品的移植性，尤其是当软件产品运行于多个平台，或有不同的功能配置版本时，Abstract Factory 模式可以减轻移植和发布时的压力，提高软件的复用性。

5. 实现

```
/* 抽象工厂的抽象类，由于使用 Java 语言，故定义为接口 */
```

```
public interface AbstractFactory {
        //创建产品的方法
        public AbstractProduct createProduct();
}
/* 抽象的产品接口 */
public interface AbstractProduct {
    ...
}
public class Product1 implements AbstractProduct {
        //实际的产品 1
}
public class Product2 implements AbstractProduct {
        //实际的产品 2
}

/* 在这个车间中，将创建实际产品 1 */
public class CFactory1 implements AbstractFactory {
        public AbstractProduct createProduct() {
            return new Product1();
        }
}

/* 在这个车间中，将创建实际产品 2 */
public class CFactory2 implements AbstractFactory {
        public AbstractProduct createProduct() {
            return new Product2();
        }
}
```

6. 相关讨论

在实际应用中，Abstract Factory 可以有更灵活的变化。事实上，如果仔细观察 Abstract Factory 模式就可以发现，对于 Client 来说，最关注的就是在不同条件下获得接口一致但实现不同的对象，只要避免类名的硬编码，采用其他方式也可以实现。所以，也可以采用其他的方式实现。例如，在 Java 中就可以采用接口的方式实现，如图 12-3 所示。

图 12-3 中所描绘的类就应用了 Abstract Factory 的思想，采用面向接口的方式，简单实现了工厂模式。Product Factory 既可以看作抽象工厂 Abstract Factory，也可以看作具体的工厂（车间）Concrete Factory。其中提供了一个获得产品的方法：getProduct (productName：String)，该方法将根据产品的名称创建特定的产品，并返回。所有的产品都实现了 Product 接口，可以在客户程序中加以使用。与抽象工厂模式一样，工厂中

获得对象的方法（getProduct()）与实际的产品线数量是一致的，如果要增加新的产品线，例如，定义新的产品接口 ProductX，需要增加相应的 getProduct()方法。由于这种方式把 Abstract Factory 和 Concrete Factory 合并为一个 Product Factory，所以增加新的实现 Product 接口的 Product*n* 不需要修改任何代码。

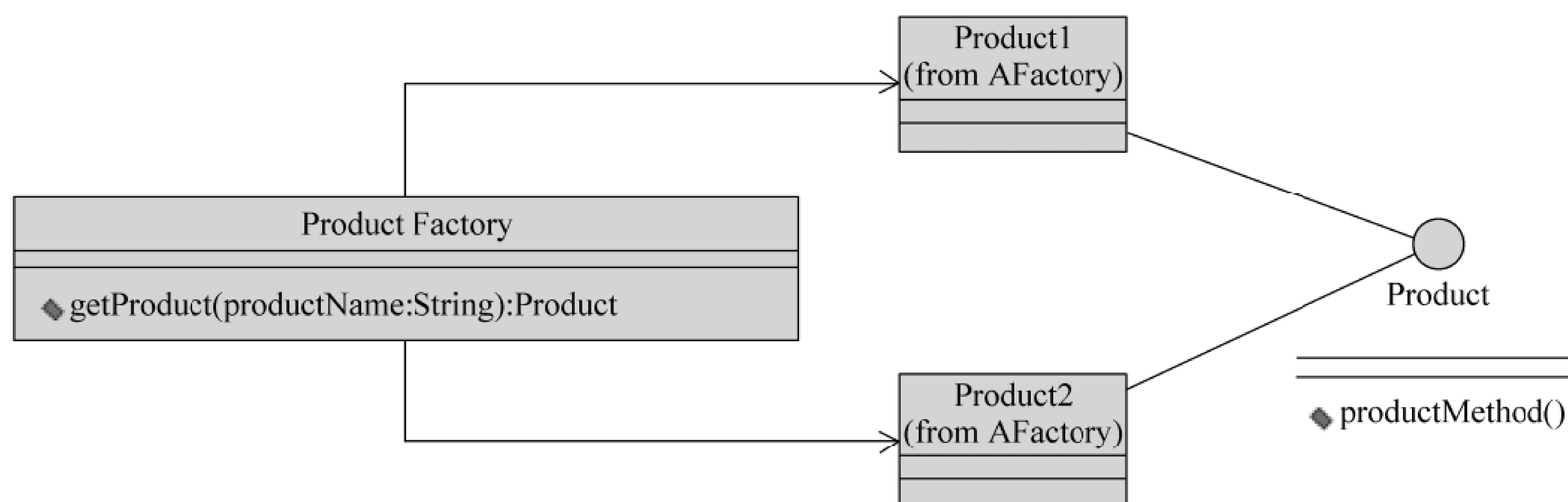


图 12-3　面向接口的简单工厂模式

除了这种面向接口的方法外，工厂模式还可以有更多的变化。学习设计模式最重要的是学习设计思想，学习了工厂模式后，应该知道：

（1）可配置的对象创建方法可以提高系统的移植性和复用性。

（2）充分利用面向对象多态的特性可以避免对象创建过程的硬编码。

12.5　MVC 架构的设计与实现

有关 MVC 的基本概念，参见 12.1 节，在本节中，具体讨论 MVC 架构的设计与实现。由于运用 MVC 的应用程序的三个部件是相互独立的，改变其中任何一个都不会影响其他两个，所以依据这种设计思想能构造良好的松耦合的构件。

12.5.1　MVC 架构

MVC 架构框架可以包括三个抽象类，一个是 View 抽象类，它从模型获得显示信息，并以特定的形式展示给用户，对于相同的信息可以有多个不同的显示形式或视图；一个是 Controller 抽象类，处理用户与软件的交互操作，其职责是决定软件的控制流程，确保用户界面与模型间的对应联系，它接受用户的输入，将输入反馈给模型，进而实现对模型的计算控制，再根据用户的需求，创建合适的视图返回到用户界面，它是使模型和视图协调工作的部件；另外一个是 Model 抽象类，它向视图（View 抽象类）和控制器（Controller 抽象类）提供业务逻辑服务。这三个抽象类之间的关系可以描述如下：控制器把接收到的请求或数据传送到模型去处理，再根据用户的要求，创建一个合适的视图，

该视图从模型中读取处理后的结果把其以特定的形式显示出来。

由此可见，控制器用于创建视图，所以，可以采用 Factory 模式，而由于控制器将输入数据反馈给模型，进而实现对模型的计算控制，所以可以采用 Bridge 模式来实现（也可以采用 Adapter 模式来实现，如果采用这种模式，Controller 和 Model 两个抽象类之间的关系就是集合关系），视图从模型获得显示信息，并以特定的形式展示给用户，同样可以采用 Bridge 模式来实现。

综上所述，整个 MVC 架构框架的 UML 图如图 12-4 所示。

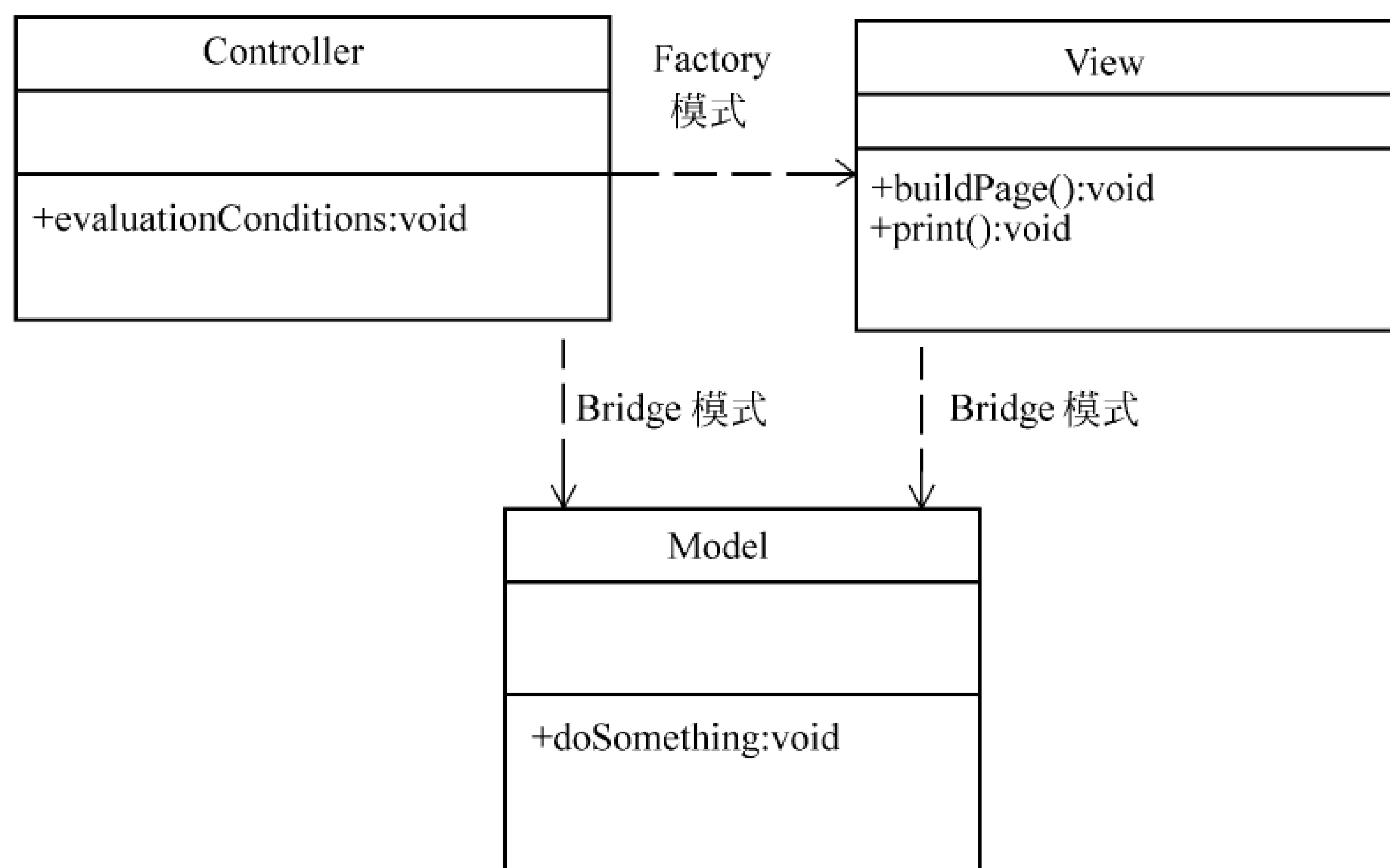


图 12-4　MVC 架构框架图

12.5.2　MVC 的设计与实现

在这一节中，以网络应用为例，讨论 MVC 架构的具体设计与实现。图 12-5 是一个简单的应用 MVC 架构的瘦客户网络应用程序的结构图。从图 12-5 中可以看出，整个系统的结构分为两个部分，一部分是客户端，另一部分是服务器。客户端不运行任何和应用程序相关的代码，它所需要的工具就是网络浏览器，通过网络浏览器，用户可以向服务器提出各种请求，包括显示网页、查询信息等请求，然后将服务器返回的请求结果再通过网络浏览器显示出来以满足自己的需求；而服务器和客户端不同，整个系统的代码都运行在服务器端，而 MVC 架构也是在服务器端实现的。

当客户端发送一个请求到服务器时，MVC 中的控制器负责接收这个请求，并将请求中包含的参数传送给模型，同时，调用模型中的服务来处理这个请求，并根据模型的处理结果选择一个合适的视图，由这个视图创建一个 HTML 页面返回给用户。在这里，如果有必要，视图可以调用模型服务来获取处理请求的结果数据，以便组建 HTML 页面供用户查看。例如，当控制器接收到一个用户登录请求时，它首先将用户名和密码传送

给模型去处理，而模型通过查询数据库，检查传送来的用户名和密码是否合法，并返回成功与否的标志。控制器根据模型返回的成功与否的标志来选取不同的视图，如果失败，则选取登录失败视图，由该视图组建一个 HTML 页面，提示用户输入的用户名或密码无效；如果成功，则选取登录成功视图，由该视图组建一个 HTML 页面，由于该页面需要显示该用户的某些个人信息，所以，该视图调用模型的服务来得到该用户的相关信息以完成页面的组建，并将其发送给用户。

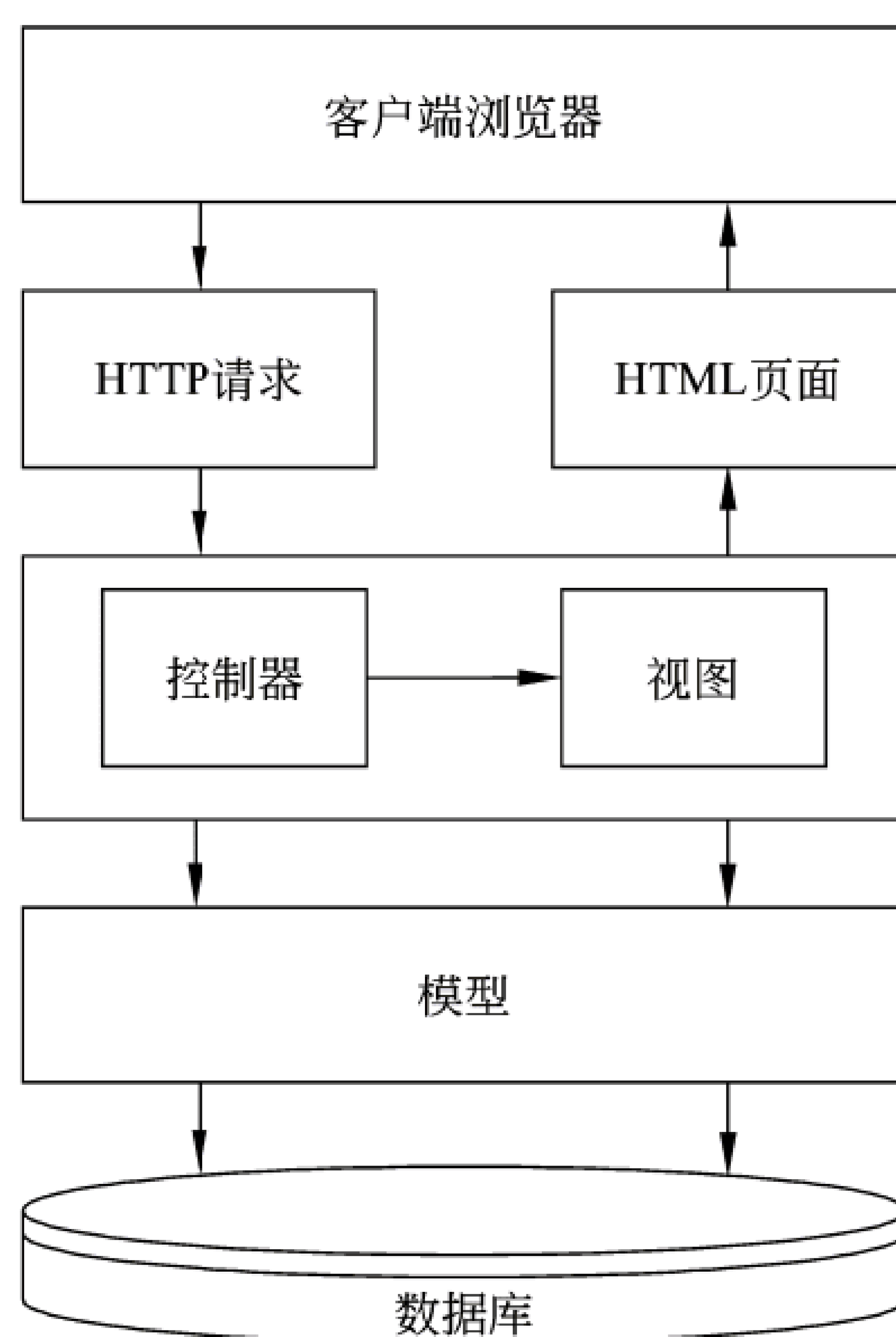


图 12-5 MVC 架构在网络中的应用

为了实现图 12-5 所描述的 MVC 网络应用程序，用下面的 UML 类图来说明其具体的设计和实现过程，如图 12-6 所示。

把用户请求封装到一个 HttpEvent 类中，这个类中的数据成员包含请求参数，例如，用户名和密码等；控制器被封装在 Controller 类中，在这个类中，包括处理用户请求的成员函数，例如，判断用户的请求是否合法或选取合适的视图等；模型被封装在 Model 类中，在这个类中包括一些业务处理逻辑的成员函数，例如，根据控制器传送的用户信息判断该用户是否合法等；视图被封装在 View 类中，这个类包含一些和显示相关的成员函数，例如，建立一个 HTML 页面或将该页面传送到客户端等。

对于大型网络应用程序，还可以扩展 MVC 架构，引进一个引擎，由这个引擎接收来自客户的请求，并根据请求的类型选取一个合适的控制器，由该控制器去处理这个请求。不难看出，这种扩展的 MVC 架构 Abstract Factory 模式的应用。

MVC 架构是创建软件的很好途径，它所提倡的一些原则，例如，内容和显示互相分离；隔离模型、视图和控制器的构件等，会使应用程序的架构更健壮，更具扩展性，

同时，它也会使软件在代码重用和架构方面上一个新的台阶。然而，MVC 由于没有明确的定义，所以完全理解 MVC 并不是很容易。使用 MVC 需要精心的计划，由于它的内部原理比较复杂，所以需要花费一些时间去思考。

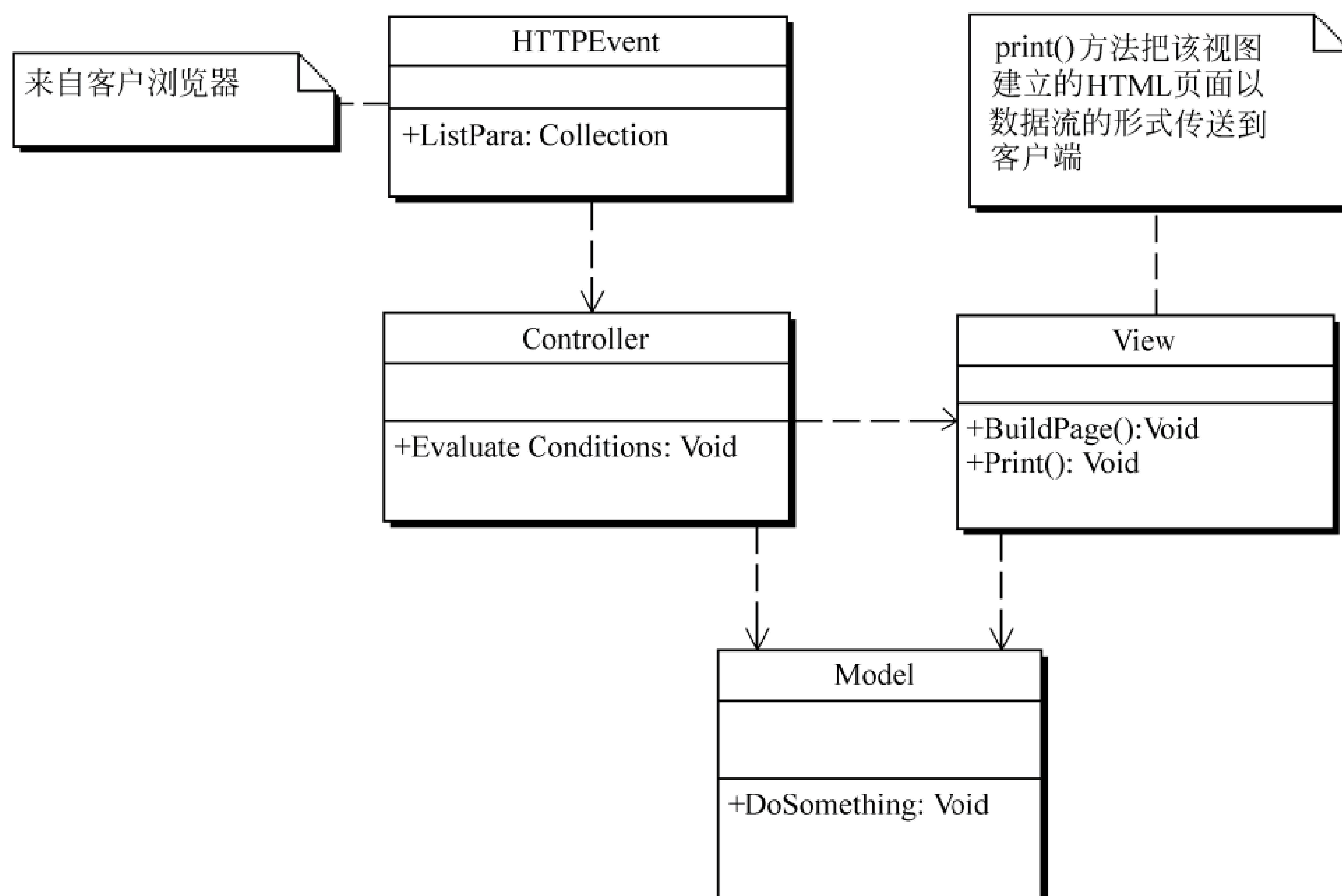


图 12-6　实现 MVC 的类图

本章参考文献

[1] 张友生. 软件体系结构（第 2 版）. 北京：清华大学出版社，2003
[2] 周之英. 现代软件工程（下）. 北京：科学出版社，2000
[3] Server-side MVC Architecture. http：//www.uidesign.net/1999/papers/webmvc_part1.html
[4] Ian Horrocks. Constructing the User Interface with Statecharts，Addison Wesley，1998
[5] 万建成，卢雷. 软件体系结构的原理、组成与应用. 北京：科学出版社，2002
[6] 张友生. 系统架构设计师教程. 北京：电子工业出版社，2005

第 13 章　统一建模语言

在 20 世纪 80 到 90 年代，面向对象（Object-Oriented，OO）的分析与设计方法获得了长足的发展，而且相关的研究也十分活跃，涌现出了一大批新的方法学。其中最著名的是 Booch 的 Booch 1993、Jacobson 的面向对象软件工程（Object-Oriented Software Engineering，OOSE）和 Rumbaugh 的对象建模技术（Object Modeling Technique，OMT）方法。而 UML 正是在融合了 Booch、Rumbaugh 和 Jacobson 方法论的基础上形成的标准建模语言。

13.1　UML 概述

UML 是用于系统的可视化建模语言，尽管它常与建模 OO 软件系统相关联，但由于其内建了大量扩展机制，还可以应用于更多的领域中，如工作流程、业务领域等。

（1）UML 是一种语言：UML 在软件领域中的地位与价值就像“1、2、3、+、–、…”等符号在数学领域中的地位一样。它为软件开发人员之间提供了一种用于交流的词汇表，一种用于软件蓝图的标准语言。

（2）UML 是一种可视化语言：UML 只是一组图形符号，它的每个符号都有明确语义，是一种直观、可视化的语言。

（3）UML 是一种可用于详细描述的语言：UML 所建的模型是精确的、无歧义和完整的，因此，适合于对所有重要的分析、设计和实现决策进行详细描述。

（4）UML 是一种构造语言：UML 虽然不是一种可视化的编程语言，但其与各种编程语言直接相连，而且有较好的映射关系，这种映射允许进行正向工程、逆向工程。

（5）UML 是一种文档化语言：它适合于建立系统架构及其所有的细节文档。

13.1.1　UML 的发展历史

面向对象的建模语言最早出现在 20 世纪 70 年代中期，而在 20 世纪 80 年代末开始进入快速发展阶段，截止到 1994 年，就从不到 10 种发展到 50 多种。而在 1994 年之后，各种方法论逐渐拉开了差距，以 Booch 提出的 Booch 方法和 Rumbaugh 提出的 OMT 成为了可视化建模语言的主导，而 Jacobson 的 Objectory 方法则成为最强有力的方法。

Booch 是面向对象方法最早的倡导者之一。他在 1984 年出版的 *Software Engineering with Ada* 一书中就描述了面向对象软件开发的基本问题。1991 年，他在 *Object-Oriented Design* 一书中，将以前针对 Ada 的工作扩展到整个面向对象设计领域，他对继承和类的

阐述特别值得借鉴。Booch1993 比较适合于系统的设计和构造。

Runbaugh 等人提出了 OMT，采用了面向对象的概念并引入各种独立于程序设计语言的表示符号。这种方法用对象模型、动态模型、功能模型和用例模型共同完成对整个系统的建模，所定义的概念和符号可用于软件开发的分析、设计和实现的全过程，软件开发人员不必在开发过程的不同阶段进行概念和符号的转换。OMT-2 特别适用于分析和描述以数据为中心的信息系统。

Jacobson 于 1994 年提出了 OOSE 方法，其最大特点是面向用例，在用例的描述中引入了外部角色的概念。用例的概念贯穿于整个开发过程（包括对系统的测试和验证），是精确描述需求的重要武器。目前，在学术界和工业界已普遍接受用例的概念，并认为是面向对象技术走向第二代的标志。OOSE 比较合适支持业务工程和需求分析。

各种建模语言各有长短，面对众多的建模语言，用户无力区分不同建模语言之间的差别和使用范围。由于不同的用户使用不同的建模语言和不同建模语言表达方式上的差异，使得用户之间的沟通方面出现了困难。要解决以上问题，就必须在综合分析各种不同建模语言的优缺点、适用情况的基础上统一各种不同的建模语言。

1994 年 10 月，Booch 和 Rumbaugh 开始了这项工作。首先，他们将 Booch93 和 OMT-2 统一起来，并于 1995 年 10 月发布了第一个公开版本，称之为 UM0.8（Unified Method，标准方法）。同年，Jacobson 加盟到这项工作中，经过 Booch、Rumbaugh 和 Jacobson 三人的共同努力，于 1996 年 6 月和 10 月分别发布了两个新的版本（UML0.9 和 UML0.91），并将 UM 重新命名为 UML。

1997 年，OMG 采纳了 UML，一个开放的 OO 可视化建模语言工业标准诞生了。现在 UML 已经经历了 1.1、1.2、1.4 三个版权的演变，2003 年，已经发布了最新的 2.0 版标准。

13.1.2　UML 的应用领域

（1）UML 统一了 Booch、OMT、OOSE 和其他面向对象方法的基本概念和符号，同时汇集了面向对象领域中很多人的思想，是优秀的面向对象方法和丰富的计算机科学实践中总结而成的。

（2）目前 UML 是最先进、实用的标准建模语言，而且还在不断发展进化之中。

（3）UML 是一种建模语言而不是一种方法，其中并不包括过程的概念，其本身是独立于过程的，你可以在使用过程中使用它。不过与 UML 结合最好的是用例驱动的、以体系结构为中心的、迭代的、增量的开发过程。

作为一种标准建模语言，UML 的核心是以面向对象的思想来描述客观世界，具有广阔的应用领域。目前主要应用领域是在软件系统建模，但是它同样可以应用于其他领域，如机械系统、企业机构或业务过程，以及处理复杂数据的信息系统、具有实时要求的工业系统或工业过程等。总之，UML 是一个通用的标准建模语言，可以对任何系统的

动态行为和静态行为进行建模。同时，标准建模语言 UML 可以对信息系统提供从需求分析到系统维护的整个生命周期提供有效的支持。在需求分析阶段，可以通过用例模型来捕获和组织用户的需求。分析系统对于用户的价值。通过用例建模，描述对系统感兴趣的外部角色及其对系统（用例）的功能要求。分析阶段主要关心问题域中的主要概念（如抽象、类和对象等）和机制，以及这些概念之间的相互协作，并用 UML 类图来描述。至于类之间的协作关系则可以用交互图和顺序图来描述。在分析阶段，只对问题域的对象（现实世界的概念）建模，而不考虑定义软件系统中技术细节的类（如处理用户接口、数据库、通信和并行性等问题的类）。这些技术细节将在设计阶段引入，因此设计阶段为构造阶段提供更详细的规格说明。

编码阶段的主要任务是将设计阶段设计结果转换成为实际的代码。在设计阶段需要注意的是不要过早的考虑设计结果要用哪种编程语言实现。如果过早的考虑这个问题，会使设计工作陷入细节的泥潭，不利于对模型进行全面理解。

标准建模语言 UML 还可以对测试阶段提供有效的支持。不同的测试阶段可以使用不同的 UML 图作为测试的依据。比如，在单元测试阶段，可以使用类图和类的规格说明来制导测试；在集成阶段，可以使用合作图、活动图和部署图；系统测试和验收测试阶段则可以使用顺序图和用例图来验证系统的外部行为。

总之，标准建模语言 UML 能够用面向对象的方法描述任何类型的系统，并对系统开发从需求调研到测试和维护的各个阶段进行有效的支持。

13.2 UML 的结构

UML 的结构包括 UML 的基本构造块、支配这些构造块如何放在一起的规则（架构）和一些运用于整个 UML 的机制。

13.2.1 结构概述

本节将详细介绍 UML 结构的各组成元素。

1．构造块

UML 有三种基本的构造块，分别是事物（thing）、关系（relationship）和图（diagram）。事物是 UML 中重要的组成部分，关系把事物紧密联系在一起，图是很多有相互相关的事物的组。

2．公共机制

公共机制是指达到特定目标的公共 UML 方法，主要包括规格说明（详细说明）、修饰、公共分类（通用划分）和扩展机制 4 种。

（1）规格说明：规格说明是元素语义的文本描述，它是模型真正的核心。

（2）修饰：UML 为每一个事物设置了一个简单的记号，还可以通过修饰来表达更

多的信息。

（3）公共分类：包括类元与实体（类元表示概念，而实体表示具体的实体）、接口和实现（接口用来定义契约，而实现就是具体的内容）两组公共分类。

（4）扩展机制：包括约束（添加新规则来扩展事物的语义）、构造型（用于定义新的事物）、标记值（添加新的特殊信息来扩展事物的规格说明）。

3．规则

UML 用于描述事物的语义规则分别是为事物、关系和图命名。给一个名字以特定含义的语境，即范围；怎样使用或看见名字，即可见性；事物如何正确、一致地相互联系，即完整性；运行或模拟动态模型的含义是什么，即执行。

UML 对系统体系结构的定义是系统的组织结构，包括系统分解的组成部分、它们的关联性、交互、机制和指导原则，这些提供系统设计的信息。而具体来说，就是指 5 个系统视图，分别是逻辑视图、进程视图、实现视图、部署视图和用例视图。

（1）逻辑视图：以问题域的语汇组成的类和对象集合。

（2）进程视图：可执行线程和进程作为活动类的建模，它是逻辑视图的一次执行实例，描绘了所设计的并发与同步结构。

（3）实现视图：对组成基于系统的物理代码的文件和构件进行建模。

（4）部署视图：把构件物理地部署到一物理的、可计算的节点上，表示软件到硬件的映射及分布结构。

（5）用例视图：最基本的需求分析模型。

希赛教育专家提示：UML 还允许在一定的阶段隐藏模型的某些元素、遗漏某些元素以及不保证模型的完整性，但模型逐步地要达到完整和一致。

13.2.2　事物

UML 中的事物也称为建模元素，包括结构事物（structural things）、行为事物（behavioral things，动作事物）、分组事物（grouping things）和注释事物（annotational things，注解事物）。这些事物是 UML 模型中最基本的面向对象的构造块。

（1）结构事物。结构事物在模型中属于最静态的部分，代表概念上等或物理上的元素。总共有 7 种结构事物：

首先是类，类是描述具有相同属性、方法、关系和语义的对象的集合。一个类实现一个或多个接口。

第二种是接口，接口是指类或构件提供特定服务的一组操作的集合。因此，一个接口描述了类或构件的对外的可见的动作。一个接口可以实现类或构件的全部动作，也可以只实现一部分。

第三种是协作，协作定义了交互的操作，是一些角色和其他元素一起工作，提供一些合作的动作，这些动作比元素的总和要大。因此，协作具有结构化、动作化、维的特

性。一个给定的类可能是几个协作的组成部分。这些协作代表构成系统的模式的实现。

第四种是用例，用例是描述一系列的动作，这些动作是系统对一个特定角色执行，产生值得注意的结果的值。在模型中用例通常用来组织行为事物。用例是通过协作来实现的。

第五种是活动类，活动类是这种类，它的对象有一个或多个进程或线程。活动类和类很相似，只是它的对象代表的元素的行为和其他的元素是同时存在的。

第六种是构件，构件是物理上或可替换的系统部分，它实现了一个接口集合。在一个系统中，可能会遇到不同种类的构件。

第七种是节点，节点是一个物理元素，它在运行时存在，代表一个可计算的资源，通常占用一些内存和具有处理能力。一个构件集合一般来说位于一个节点，但有可能从一个节点转到另一个节点。

（2）行为事物：行为事物是 UML 模型中的动态部分。它们是模型的动词，代表时间和空间上的动作。总共有两种主要的行为事物。

第一种是交互（内部活动），交互是由一组对象之间在特定上下文中，为达到特定的目的而进行的一系列消息交换而组成的动作。交互中组成动作的对象的每个操作都要详细列出，包括消息、动作次序（消息产生的动作）、连接（对象之间的连接）。

第二种是状态机，状态机由一系列对象的状态组成。

内部活动和状态机是 UML 模型中最基本的两个动态事物，它们通常和其他的结构事物、主要的类、对象连接在一起。

（3）分组事物。分组事物是 UML 模型中组织的部分，可以把它们看成是个盒子，模型可以在其中被分解。总共只有一种分组事物，称为包。包是一种将有组织的元素分组的机制。结构事物、行为事物甚至其他的分组事物都有可能放在一个包中。与构件（存在于运行时）不同的是包纯粹是一种概念上的东西，只存在于开发阶段。

（4）注释事物。注释事物是 UML 模型的解释部分。

13.2.3 关系

UML 用关系把事物结合在一起，UML 中的关系主要有 4 种。

（1）依赖（dependencies）：两个事物之间的语义关系，其中一个事物发生变化会影响另一个事物的语义。

（2）关联（association）：一种描述一组对象之间连接的结构关系，如聚合关系（描述了整体和部分间的结构关系）。

（3）泛化（generalization）：一种一般化和特殊化的关系，描述特殊元素的对象可替换一般元素的对象。

（4）实现（realization）：类之间的语义关系，其中的一个类指定了由另一个类保证执行的契约。

1．用例之间的关系

两个用例之间的关系可以概括为两种情况。一种是用于重用的包含关系，用构造型<<include>>或<<use>>表示；另一种是用于分离出不同行为的扩展关系，用构造型<<extend>>表示。

（1）包含关系：当可以从两个或两个以上的原始用例中提取公共行为，或发现能够使用一个构件来实现某一个用例很重要的部分功能时，应该使用包含关系来表示它们。其中这个提取出来的公共用例称为抽象用例。

例如，希赛图书订单处理系统中，“创建新订单”和“更新订单”两个用例都需要检查客户的账号是否正确，为此定义一个抽象用例“核查客户账户”。用例“创建新订单”和“更新订单”与用例“核查客户账户”之间的关系就是包含关系。

（2）扩展关系：如果一个用例明显地混合了两种或两种以上的不同场景，即根据情况可能发生多种事情，则可以断定将这个用例分为一个主用例和一个或多个辅用例进行描述可能更加清晰。

例如，希赛图书管理系统中，读者归还图书时，需要判断当前日期是否已经超过了图书借阅的周期。如果超过了借阅周期，则必须罚款。“归还图书”和“罚款”用例之间的关系就是扩展关系。

希赛教育专家提示：用例之间还存在一种泛化关系。用例可以被特别列举为一个或多个子用例，这被称做用例泛化。当父用例能够被使用时，任何子用例也可以被使用。例如，购买飞机票时，既可以通过电话订票，也可以通过网上订票，则订票用例就是电话订票和网上订票的泛化。

2．类之间的关系

在建立抽象模型时，很少有类会单独存在，大多数都将会以某种方式彼此通信。因此，还需要描述这些类之间的关系。

（1）关联关系：描述了给定类的单独对象之间语义上的连接。关联提供了不同类之间的对象可以相互作用的连接。其余的关系涉及类元自身的描述，而不是它们的实例。用“ ———— ”表示。

（2）依赖关系。有两个元素 X，Y，如果修改元素 X 的定义可能会引起对另一个元素 Y 的定义的修改，则称元素 Y 依赖于元素 X。在 UML 中，使用带箭头的虚线“------▶”表示依赖关系。

在类中，依赖由各种原因引起，例如，一个类向另一个类发送消息；一个类是另一个类的数据成员；一个类是另一个类的某个操作参数。如果一个类的接口改变，则它发出的任何消息都可能不再合法。

（3）泛化关系。泛化关系描述了一般事物与该事物中的特殊种类之间的关系，也就是父类与子类之间的关系。继承关系是泛化关系的反关系，也就是说子类是从父类继承的，而父类则是子类的泛化。在 UML 中，使用带空心箭头的实线“————▷”表

示泛化关系，箭头指向父类。

（4）聚合关系。聚合是一种特殊形式的关联，它是传递和反对称的。聚合表示类之间的关系是整体与部分的关系。例如一辆轿车包含4个车轮、一个方向盘、一个发动机和一个底盘，就是聚合的一个例子。在UML中，使用一个带空心菱形的实线“——◇”表示聚合关系，空心菱形指向的是代表“整体”的类。

（5）组合关系。如果聚合关系中的表示“部分”的类的存在与否，与表示“整体”的类有着紧密的关系，例如“公司”与“部门”之间的关系，那么就应该使用“组合”关系来表示这种关系。在UML中，使用带有实心菱形的实线“——◆”表示组合关系。

（6）实现关系将说明和实现联系起来。接口是对行为而非实现的说明，而类之中则包含了实现的结构。一个或多个类可以实现一个接口，而每个类分别实现接口中的操作。实现关系用“------▷”表示。

（7）流关系将一个对象的两个版本以连续的方式连接起来。它表示一个对象的值、状态和位置的转换。流关系可以将类元角色在一次相互作用中连接起来。流的种类包括变成（同一个对象的不同版本）和复制（从现有对象创造出一个新的对象）两种。用“------>”表示。

希赛教育专家提示：对于聚合关系和组合关系，各种文献的说法有些区别。在这些文献中，首先定义聚集关系（整体与部分的关系），然后再把聚集关系分为两种，分别是组合聚集（相当于上述的“组合关系”）和共享聚集（相当于上述的“聚合关系”）。

13.2.4　图形

UML 2.0包括14种图，分别列举如下。

（1）类图（Class Diagram）：描述一组类、接口、协作和它们之间的关系。在面向对象系统的建模中，最常见的图就是类图。类图给出了系统的静态设计视图，活动类的类图给出了系统的静态进程视图。

（2）对象图（Object Diagram）：描述一组对象及它们之间的关系。对象图描述了在类图中所建立的事物实例的静态快照。和类图一样，这些图给出系统的静态设计视图或静态进程视图，但它们是从真实案例或原型案例的角度建立的。

（3）构件图（Component Diagram）：描述一个封装的类和它的接口、端口，以及由内嵌的构件和连接件构成的内部结构。构件图用于表示系统的静态设计实现视图。对于由小的部件构建大的系统来说，构件图是很重要的。构件图是类图的变体。

（4）组合结构图（Composite Structure Diagram）：描述结构化类（例如构件或类）的内部结构，包括结构化类与系统其余部分的交互点。它显示联合执行包含结构化类的行为的构件配置。组合结构图用于画出结构化类的内部内容。

（5）用例图（Use Case Diagram）：描述一组用例、参与者（一种特殊的类）及它们

之间的关系。用例图给出系统的静态用例视图。这些图在对系统的行为进行组织和建模时是非常重要的。

（6）顺序图（Sequence Diagram，序列图）：是一种交互图（interaction diagram），交互图展现了一种交互，它由一组对象或角色以及它们之间可能发送的消息构成。交互图专注于系统的动态视图。顺序图是强调消息的时间次序的交互图。

（7）通信图（Communication Diagram）：也是一种交互图，它强调收发消息的对象或角色的结构组织。顺序图和通信图表达了类似的基本概念，但每种图所强调的概念不同，顺序图强调的是时序，通信图则强调消息流经的数据结构。在 UML 1.X 版本中，通信图被称为协作图（Cooperation Diagram）。

（8）定时图（Timing Diagram，计时图）：也是一种交互图，它强调消息跨越不同对象或角色的实际时间，而不仅仅只是关心消息的相对顺序。

（9）状态图（State Diagram）：描述一个状态机，它由状态、转移、事件和活动组成。状态图给出了对象的动态视图。它对于接口、类或协作的行为建模尤为重要，而且它强调事件导致的对象行为，这非常有助于对反应式系统建模。

（10）活动图（Activity Diagram）：将进程或其他计算的结构展示为计算内部一步步的控制流和数据流。活动图专注于系统的动态视图。它对系统的功能建模特别重要，并强调对象间的控制流程。

（11）部署图（Deployment Diagram）：描述对运行时的处理节点及在其中生存的构件的配置。部署图给出了体系结构的静态部署视图，通常一个节点包含一个或多个部署图。

（12）制品图（Artifact Diagram）：描述计算机中一个系统的物理结构。制品包括文件、数据库和类似的物理比特集合。制品图通常与部署图一起使用。制品也给出了它们实现的类和构件。

（13）包图（Package Diagram）：描述由模型本身分解而成的组织单元，以及它们的依赖关系。

（14）交互概览图（Interaction Overview Diagram）：是活动图和顺序图的混合物。

13.3 用例图

用例实例是在系统中执行的一系列动作，这些动作将生成特定参与者可见的价值结果。一个用例定义一组用例实例。它确定了一个和系统参与者进行交互、并可由系统执行的动作序列。用例模型描述的是外部执行者（actor）所理解的系统功能。用例模型用于需求分析阶段，它的建立是系统开发者和用户反复讨论的结果，表明了开发者和用户对需求规格达成的共识。

在 UML 中，用例表示为一个椭圆。图 13-1 显示了希赛教育图书管理系统的用例图。

其中，“新增书籍信息”、“查询书籍信息”、“修改书籍信息”、“登记外借情况”、“查询外借情况”、“统计金额与册数”等都是用例的实例。

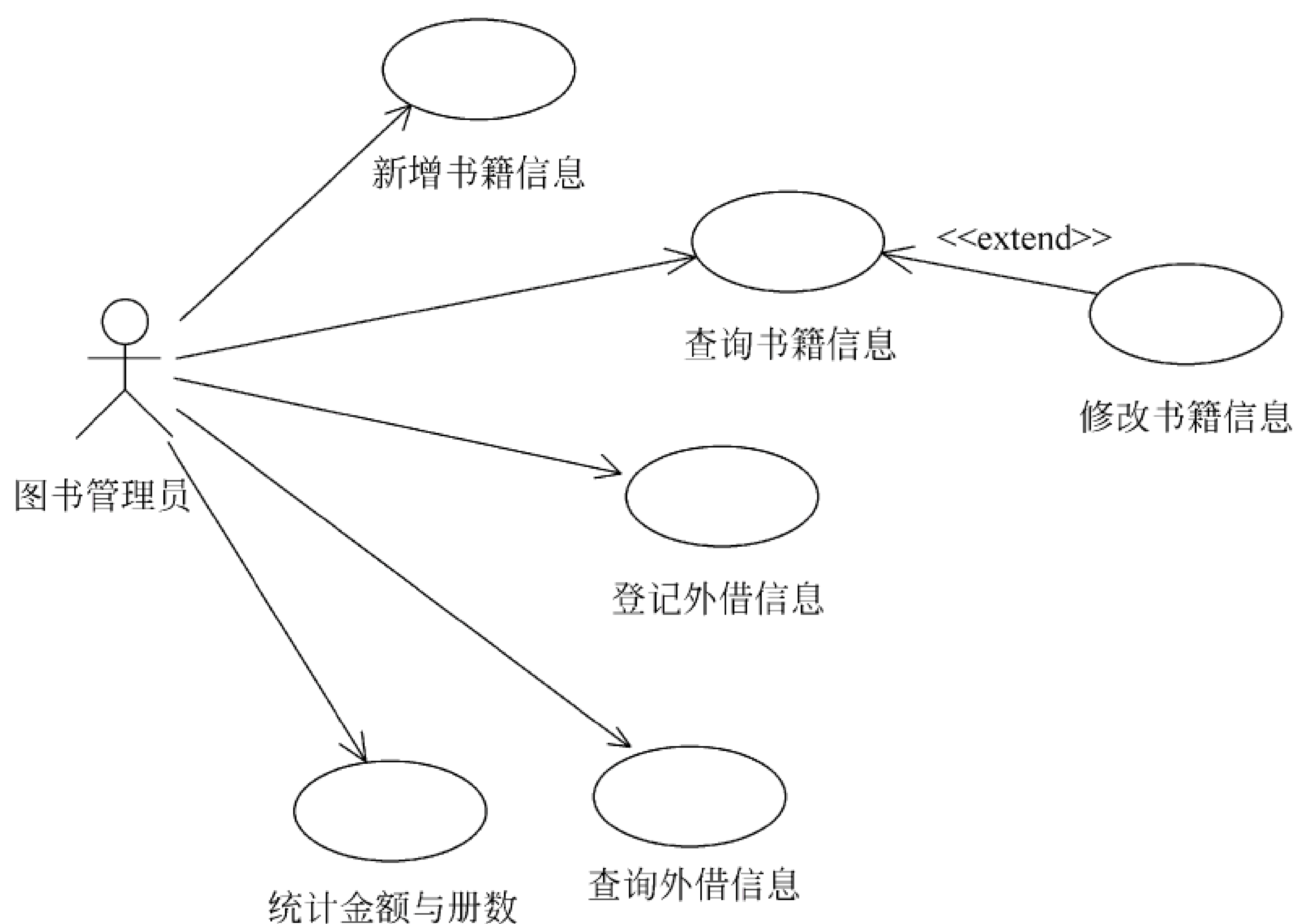


图 13-1　用例图示例

1．参与者

参与者代表与系统接口的任何事物或人（包括其他系统），它是指代表某一种特定功能的角色，因此参与者都是虚拟的概念。在 UML 中，用一个小人表示参与者。

图 13-1 中的“图书管理员”就是参与者。对于该系统来说，可能可以充当图书管理员角色的有多个人，因为他们对于系统而言均起着相同的作用，扮演相同的角色，因此，只使用一个参与者表示。切忌不要为每一个可能与系统交互的真人画出一个参与者。

可以通过以下问题来帮助寻找到系统的相关参与者：

- 谁是系统的主要用户？
- 谁从系统获得信息？
- 谁向系统提供信息？
- 谁从系统删除信息？
- 谁支付、维护系统？
- 谁管理系统？
- 系统需要与其他哪些系统交互？
- 系统需要操纵哪些硬件？
- 在预设的时间内，有事情自动发生吗？
- 系统从哪里获得信息？

- 谁对系统的特定需求感兴趣？
- 几个人在扮演同样的角色吗？
- 一个人扮演几个不同的角色吗？
- 系统使用外部资源吗？
- 系统用在什么地方？

2．用例

用例是对系统行为的动态描述，它可以促进设计人员、开发人员与用户的沟通，理解正确的需求，还可以划分系统与外部实体的界限，是系统设计的起点。在识别出参与者之后，可以使用下列问题帮助识别用例：

- 每个参与者的任务是什么？
- 有参与者将要创建、存储、修改、删除或读取系统中的信息吗？
- 什么用例会创建、存储、修改、删除或读取这个信息？
- 参与者需要通知系统外部的突然变化吗？
- 需要把系统中正在发生的事情通知参与者吗？
- 什么用例将支持和维护系统？
- 所有的功能需求都对应到用例中了吗？
- 系统需要何种输入输出？输入从何处来？输出到何处？
- 当前运行系统的主要问题是什么？

13.4　类图和对象图

在面向对象建模技术中，对象是指现实世界中有意义的事物具有封装性和自治性的特点，而类是指具有相同属性和行为的一组对象。类、对象和它们之间的关联是面向对象技术中最基本的元素。对于一个想要描述的系统，其类模型和对象模型揭示了系统的结构。在 UML 中，类和对象模型分别由类图和对象图表示。类图技术是 OO 方法的核心。图 13-2 显示了希赛教育图书管理系统的类图。

对象与我们对客观世界的理解相关，它通常用来描述客观世界中某个具体的事物。而类是对一组具有相同属性，表现相同行为的对象的抽象。因此，对象是类的实例。在 UML 中，类的可视化表示为一个划分成三个格子的长方形（下面两个格子可省略）。在图 13-2 中，“书籍”、“借阅记录”等都是一个类。

（1）类的命名：最顶部的格子包含类的名字。类的命名应尽量用应用领域中的术语，应明确、无歧义，以利于开发人员与用户之间的相互理解和交流。

（2）类的属性：中间的格子包含类的属性，用以描述该类对象的共同特点。该项可省略。在图 13-2 中，“书籍”类有“书名”、“书号”等属性。UML 规定类的属性的语法

为："可见性 属性名：类型 ＝ 默认值{约束特性}"。

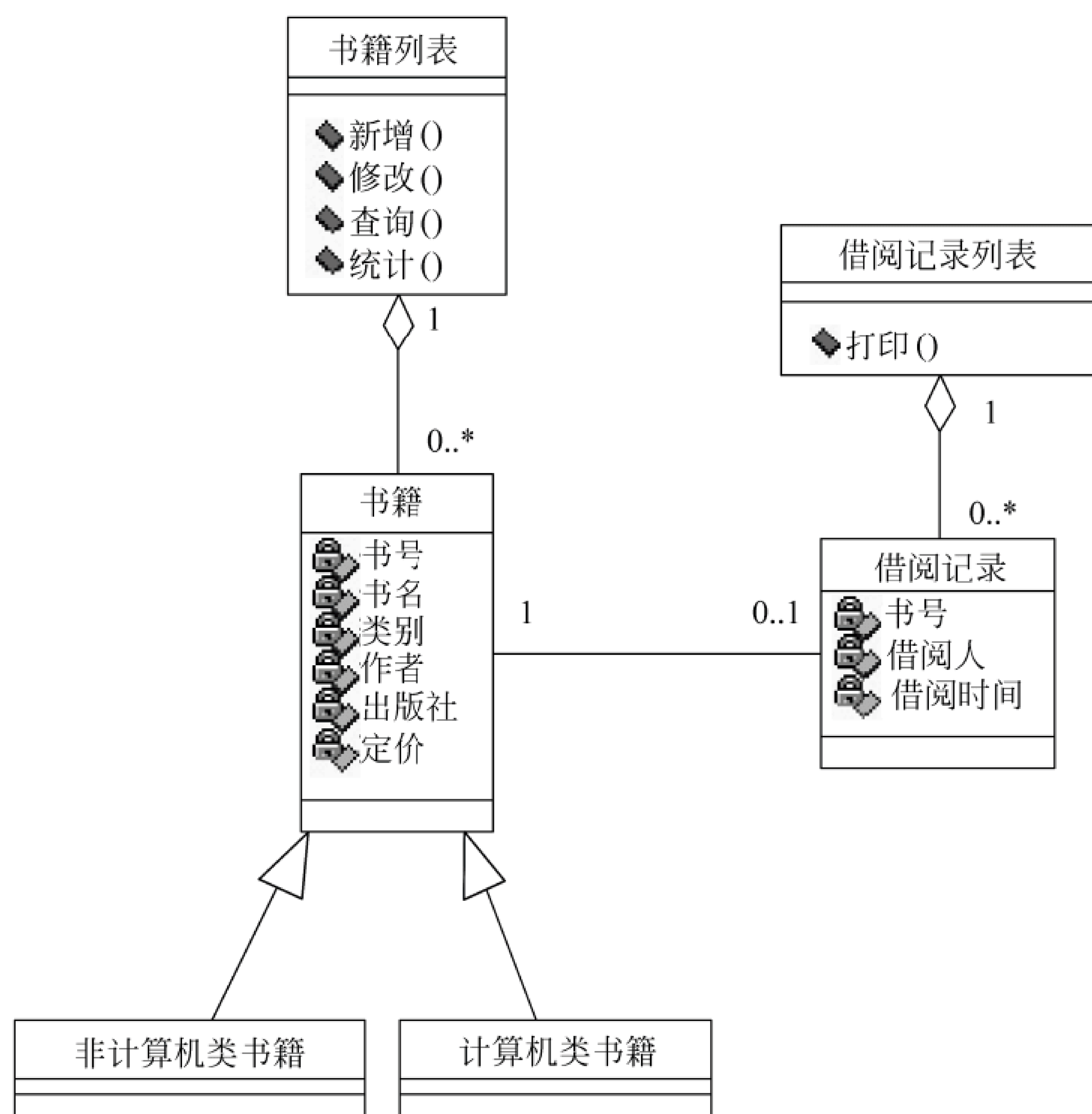


图 13-2 希赛教育图书管理系统类图

- 可见性：包括 Public、Private 和 Protected，分别用+、–、#号表示。
- 类型：表示该属性的种类，它可以是基本数据类型（如整数、实数、布尔型等），也可以是用户自定义的类型。一般它由所涉及的程序设计语言确定。
- 约束特性：用户对该属性性质有个约束的说明。例如，"{只读}"说明该属性是只读属性。

（3）类的操作（operation）：该项可省略。操作用于修改、检索类的属性或执行某些动作。操作通常也被称为功能，但是它们被约束在类的内部，只能作用到该类的对象上。操作名、返回类型和参数表组成操作界面。UML 规定操作的语法为："可见性：操作名（参数表）：返回类型 {约束特性}"。

类图描述了类和类之间的静态关系。定义了类之后，就可以定义类之间的各种关系了，请参考 13.2.3 节的介绍。类图描述类和类之间的静态关系，与数据模型不同，它不仅显示了信息的结构，同时还描述了系统的行为。类图是面向对象建模中最重要的模型。

UML 中对象图与类图具有相同的表示形式，对象图可以看作是类图的一个实例。对象之间的链（link）是类之间的关联的实例。

13.5　交互图

交互图是表示各组对象如何依某种行为进行协作的模型。通常可以使用一个交互图来表示和说明一个用例的行为。在 UML 中，包括 4 种不同形式的交互图，分别是顺序图、通信图、定时图和交互概览图。其中，顺序图强调的是时序，通信图则强调消息流经的数据结构，定时图强调消息跨越不同对象或角色的实际时间，交互概览图是活动图和顺序图的混合物。顺序图和通信图是两种基本的交互图，它们之间没有什么本质不同，只是排版不尽相同而已；定时图和交互概览图是两种特殊的变体。

本节简单介绍顺序图、通信图和定时图，有关交互概览图的知识，请阅读 13.7.3 节。

13.5.1　顺序图

顺序图用来描述对象之间动态的交互关系，着重体现对象间消息传递的时间顺序。顺序图允许直观地表示出对象的生存期，在生存期内，对象可以对输入消息做出响应，并且可以发送信息。

正如图 13-3 所示，顺序图存在两个轴，水平轴表示不同的对象，垂直轴表示时间，表示对象及类的生命周期。

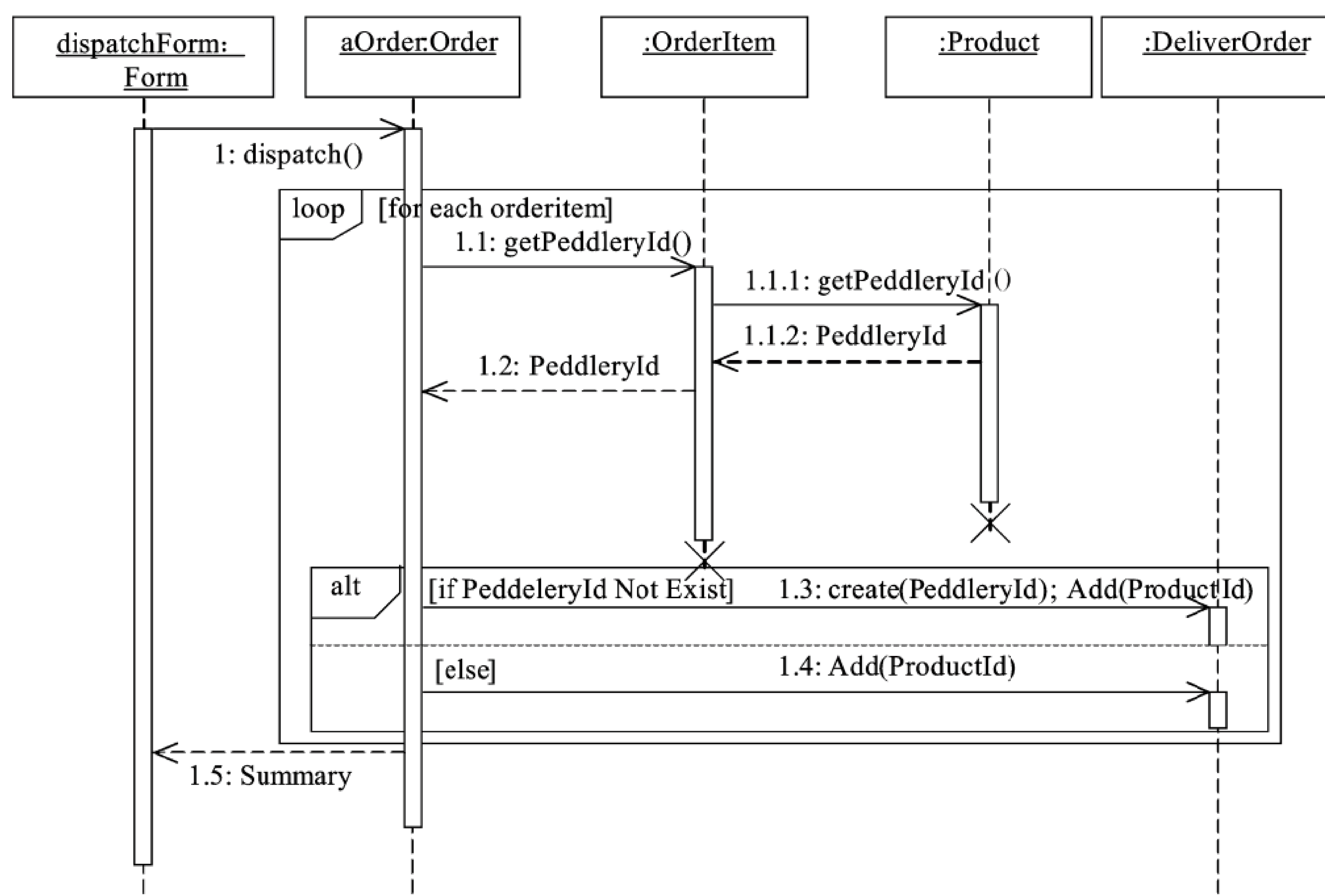


图 13-3　顺序图示例

对象间的通信通过在对象的生命线间画消息来表示。消息的箭头指明消息的类型。顺序图中的消息可以是信号、操作调用或类似于 C++中的 RPC 和 Java 中的 RMI。当收到消息时，接收对象立即开始执行活动，即对象被激活了。通过在对象生命线上显示一个细长矩形框来表示激活。

消息可以用消息名及参数来标识，消息也可带有顺序号。消息还可带有条件表达式，表示分支或决定是否发送消息。如果用于表示分支，则每个分支是相互排斥的，即在某一时刻仅可发送分支中的一个消息。

13.5.2 通信图

通信图强调收发消息的对象或角色的结构组织。顺序图和通信图表达了类似的基本概念，但每种图强调概念的不同视图，顺序图强调时序，通信图强调消息流经的数据结构。图 13-4 就是与图 13-3 相对应的通信图，可以从图 13-4 中很明显地发现通信图与顺序图之间的异同点。

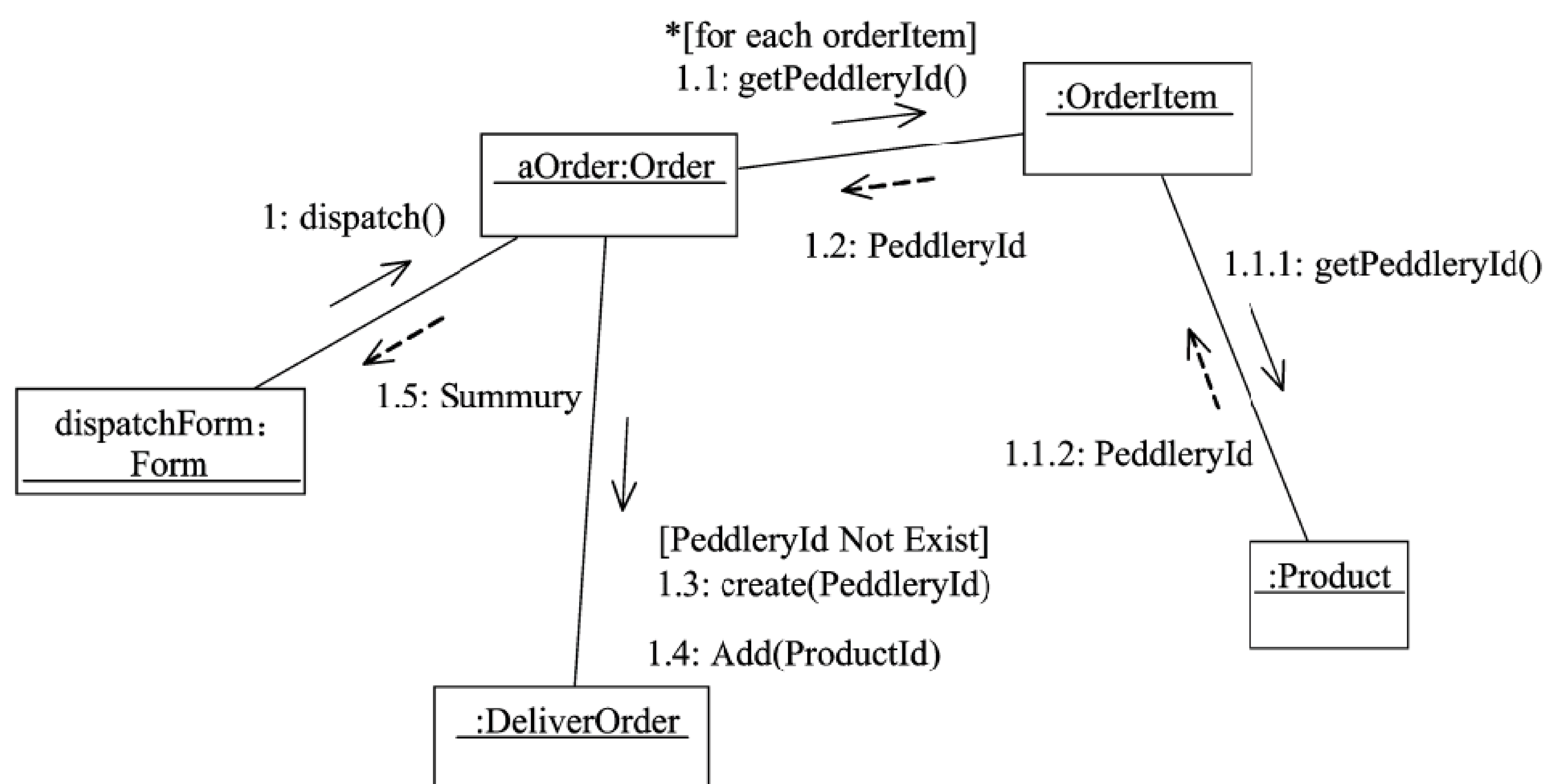


图 13-4 通信图示例

13.5.3 定时图

如果要表示的交互具有很强的时间特性（例如，现实生活中的电子工程、实时控制等系统中），在 UML 1.X 中是无法有效地表示出来的。而在 UML 2.0 中引入了一种新的交互图来解决这类问题，这就是着重表示定时约束的定时图。

根据 UML 的定义，定时图实际上是一种特殊形式的顺序图（即一种变化），它与顺序图的区别主要体现在以下几个方面：

（1）坐标轴交换了位置，改为从左到右来表示时间的推移。

（2）用生命线的“凹下凸起”来表示状态的变化，每个水平位置代表一种不同的状

态，状态的顺序可以有意义，也可以没有意义。

（3）生命线可以跟在一根线后面，在这根线上显示一些不同的状态值。

（4）可以显示一个度量时间值的标尺，用刻度来表示时间间隔。

图 13-5 是一个定时图的实际例子，其中小黑点加曲线表示的是注释。它用来表示一个电子门禁系统的控制逻辑，该门禁系统包括门（物理的门）、智能读卡器（读取用户的智能卡信息）、处理器（用来处理是否开门的判断）。

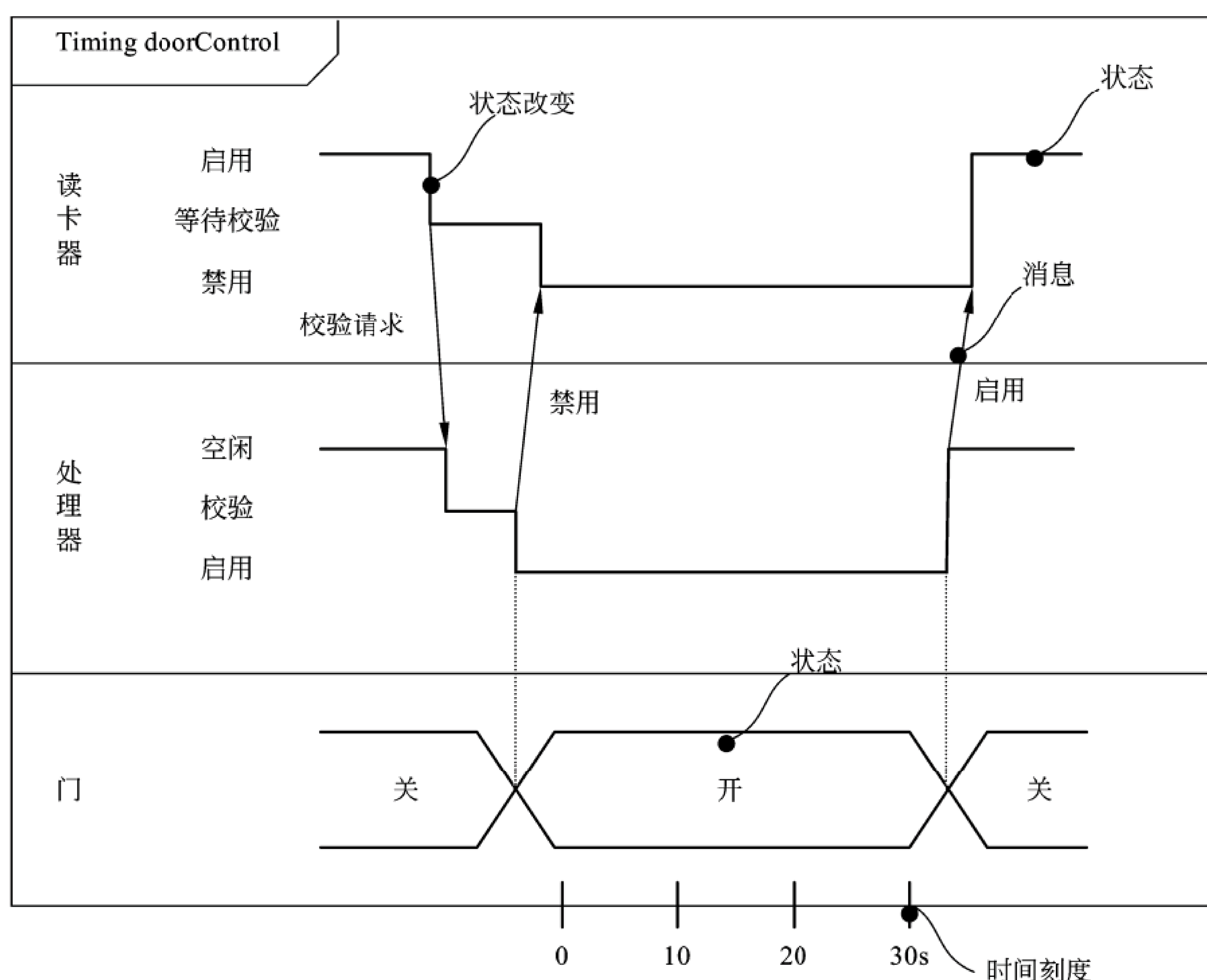


图 13-5　定时图示例

在这个例子中，它所表示的意思是一开始读卡器是启用的（等用户来刷卡）、处理器是空闲的（没有验证的请求）、门是关的；接着，当用户刷卡时，读卡器就进入了“等待校验”的状态，并发一个消息给处理器，处理器就进入了校验状态。如果校验通过，就发送一个“禁用”消息给读卡器（因为门开的时候，读卡器就可以不工作了），使读卡器进入禁用状态；并且自己转入启用状态，这时门的状态变成了“开”。而门“开”了 30 秒钟（根据时间刻度得知）之后，处理器将会把它再次“关”上，并且发送一个“启用”消息给读卡器（门关了，读卡器又要重新工作了），这时读卡器再次进入启用状态，而处理器已经又回到了空闲状态。

从上面的例子中，不难看出定时图所包含的图元并不多，主要包括生命线、状态、

状态变迁、消息、时间刻度，可以根据自身的需要来使用它。

13.6 状态图

状态图用来描述对象状态和事件之间的关系，通常用状态图来描述单个对象的行为。它确定了由事件序列引出的状态序列，但并不是所有的类都需要使用状态图来描述它的行为，只有那些具有重要交互行为的类，才会使用状态图来描述。

图 13-6 是一个数码冲印店的订单状态图实例，正如图 13-6 所示，状态图包括以下部分。

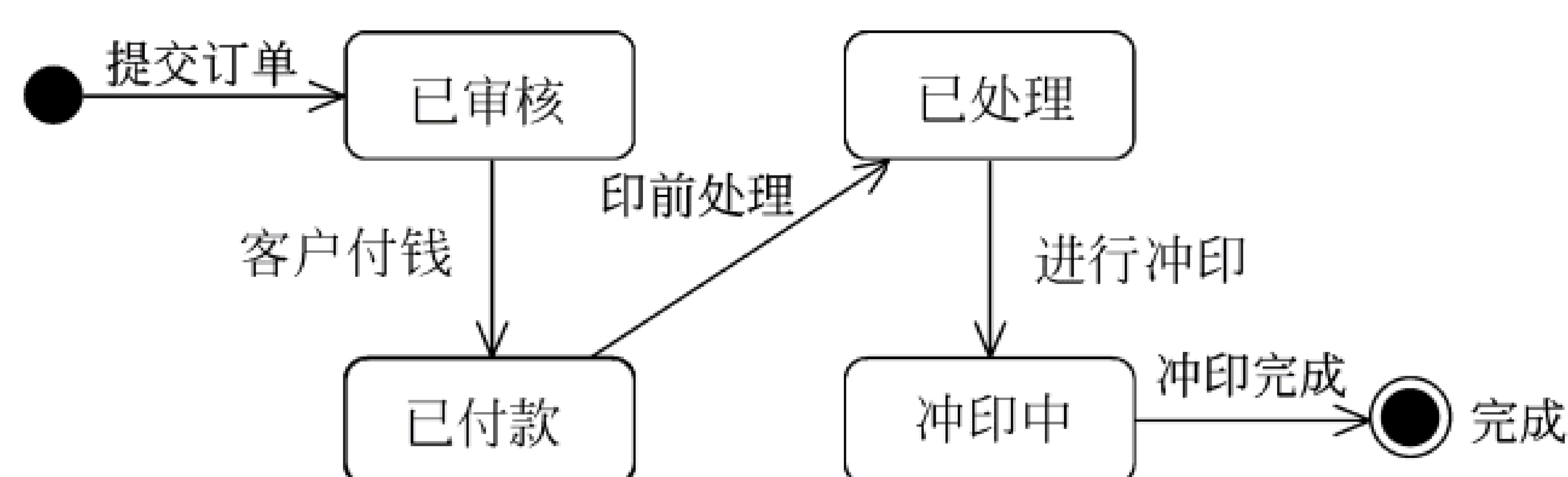


图 13-6　状态图示例

（1）状态：又称为中间状态，用圆角矩形框表示。

（2）初始状态：又称为初态，用一个黑色的实心圆圈表示，在一张状态图中只能够有一个初始状态。

（3）结束状态：又称为终态，在黑色的实心圆圈外面套上一个空间圆，在一张状态图中可能有多个结束状态。

（4）状态转移：用箭头说明状态的转移情况，并用文字说明引发这个状态变化的相应事件是什么。

一个状态也可能被细分为多个子状态，那么如果将这些子状态都描绘出来的话，则这个状态就是复合状态。

状态图适合用于表述在不同用例之间的对象行为，但并不适合于表述包括若干协作的对象行为。通常不会需要对系统中的每一个类绘制相应的状态图，而通常会在业务流程、控制对象、用户界面的设计方面使用状态图。

13.7 活动图

活动图用来表示系统中各种活动的次序，它的应用非常广泛，它既可用来描述用例的工作流程，也可以用来描述类中某个方法的操作的行为。活动图是由状态图变化而来的，它们各自用于不同的目的。活动图依据对象状态的变化来捕获动作（将要执行的工

作或活动）与动作的结果。活动图中一个活动结束后将立即进入下一个活动（在状态图中，状态的变迁可能需要事件的触发）。

13.7.1　基本活动图

图 13-7 给出了一个基本活动图的例子。

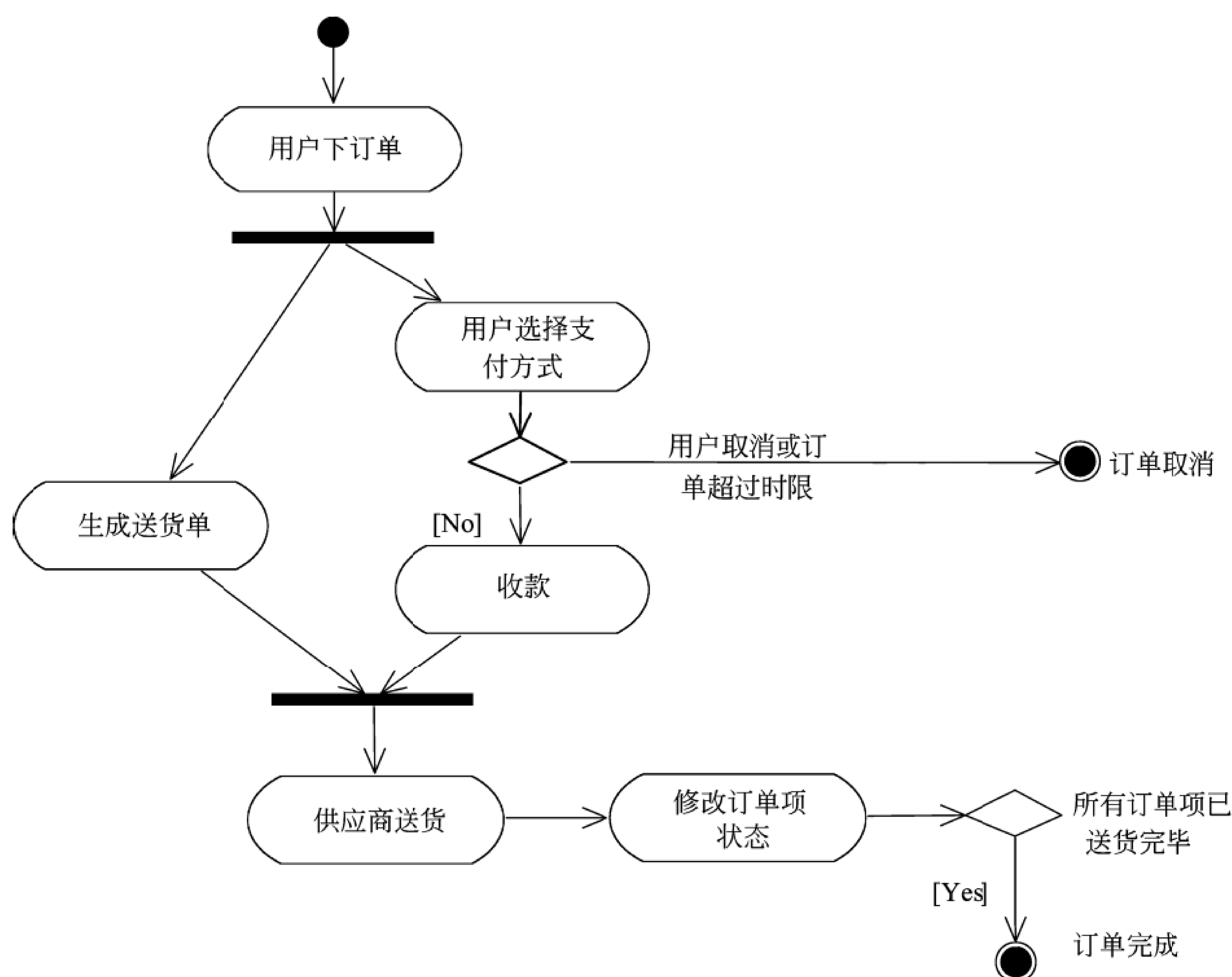


图 13-7　基本活动图示例

正如图 13-7 所示，活动图中与状态图类似，包括了初始状态、终止状态，以及中间的活动状态，每个活动之间，也就是一种状态的变迁。在活动图中，还引入了以下几个概念。

（1）判定：说明基于某些表达式的选择性路径，在 UML 中使用菱形表示。

（2）分叉与结合：由于活动图建模时经常会遇到并发流，因此，在 UML 中引入了如图 13-7 所示的粗线来表示分叉和结合。

13.7.2　带泳道的活动图

在前面说明的基本活动图中，虽然能够描述系统发生了什么，但无法说明完成这个活动的对象。而针对面向对象的程序设计（Object-Oriented Programming，OOP）而言，

这就意味着活动图没有描述出各个活动由哪个类来完成。要想解决这个问题，可以通过泳道。泳道将活动图的逻辑描述与顺序图、协作图的责任描述结合起来，如图 13-8 所示。

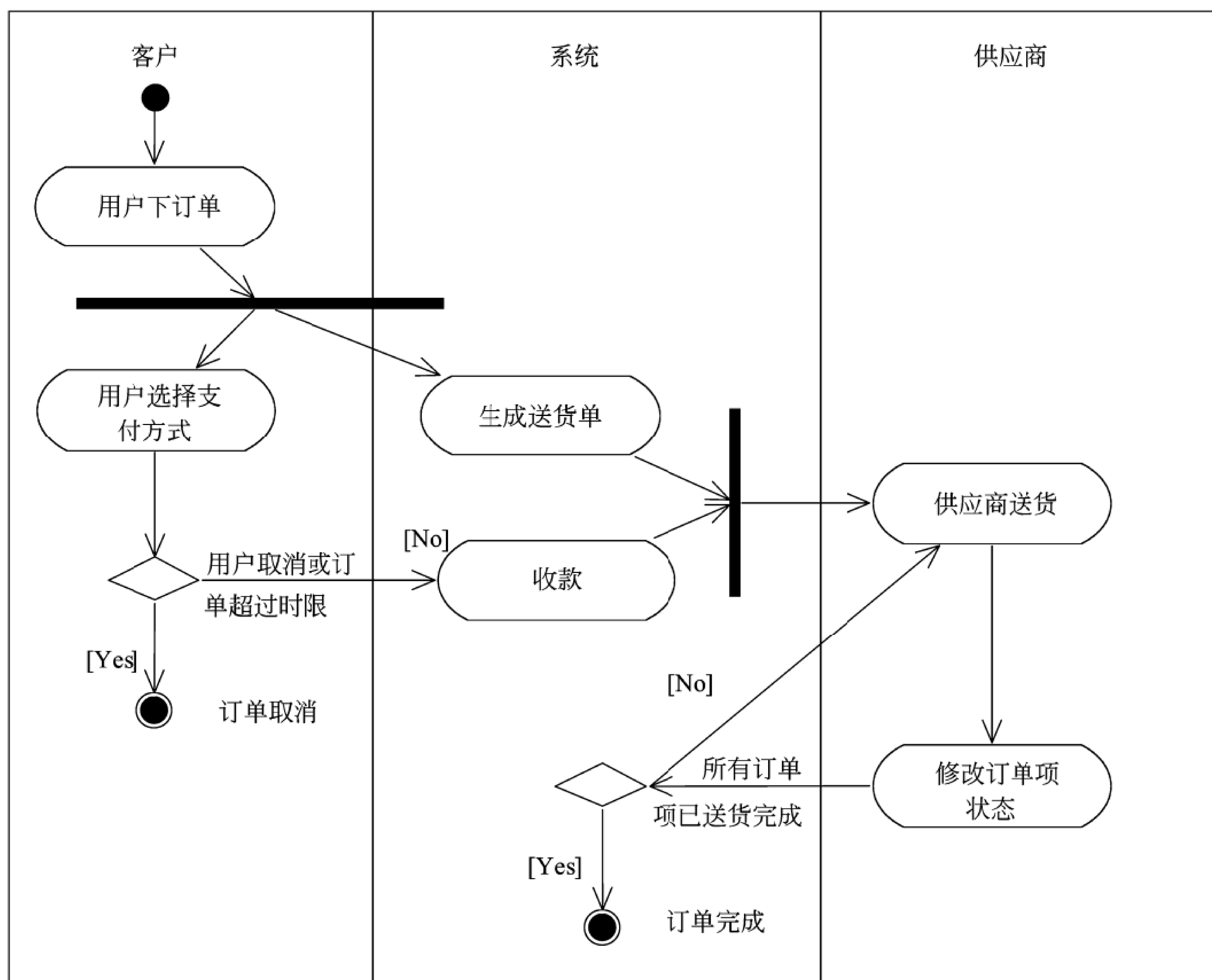


图 13-8 带泳道活动图示例

在活动图中，对象可以作为活动的输入或输出，对象与活动间的输入/输出关系由虚线箭头来表示。如果仅表示对象受到某一活动的影响，则可用不带箭头的虚线来连接对象与活动。

在活动图中，可以通过信号的发送和接收标记来表示信号的发送和接收，发送和接收标志也可与对象相连，用于表示消息的发送者和接收者。

13.7.3 交互概览图

交互概览图并没有引入新的事物，其所有的图示法都已经在顺序图和活动图中阐述过了。例如，在图 13-7 中关于“用户订单处理”的活动图中，如果想表达出“用户下订单”和“生成送货单”两个活动节点内的对象控制流，就可以结合 13.5.1 中顺序图的知

识，绘制出一张交互概览图，其结果如图 13-9 所示。

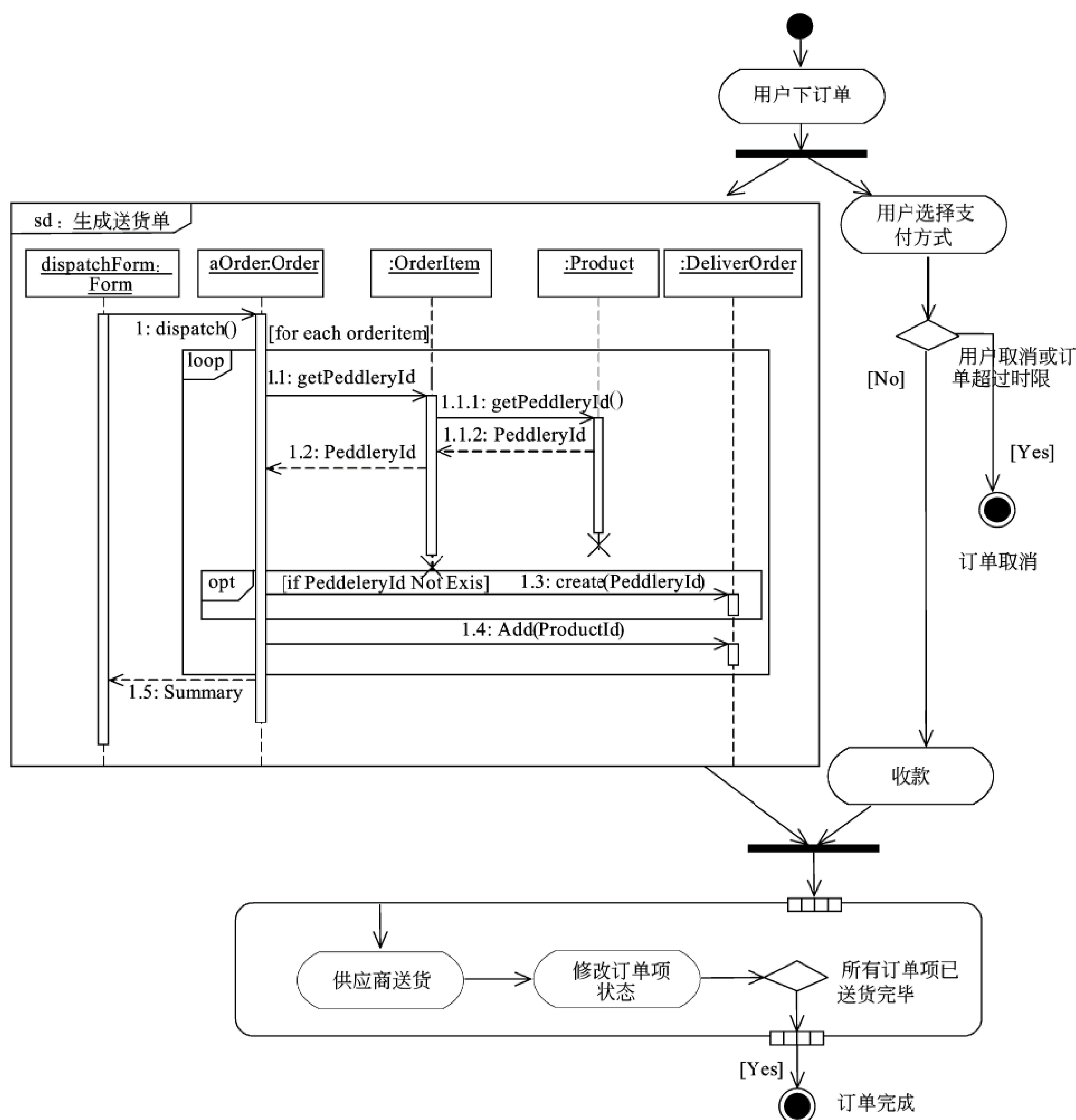


图 13-9　交互概览图示例

13.8　构件图

对面向对象系统物理方面进行建模，需要使用两种图，分别是构件图和部署图。构件图可以有效地显示一组构件，以及它们之间的关系。构件图中通常包括构件、接口以及各种关系。13-10 就是一个构件图的例子。

构件指的是源代码文件、二进制代码文件和可执行文件等，而构件图就是用来显示编译、链接或执行时构件之间的依赖关系。例如，在图 13-10 中，就是说明 QueryClient.exe

将通过调用 QueryServer.exe 来完成相应的功能，而 QueryServer.exe 则需要 Find.exe 的支持，Find.exe 在实现时调用了 Query.dll。

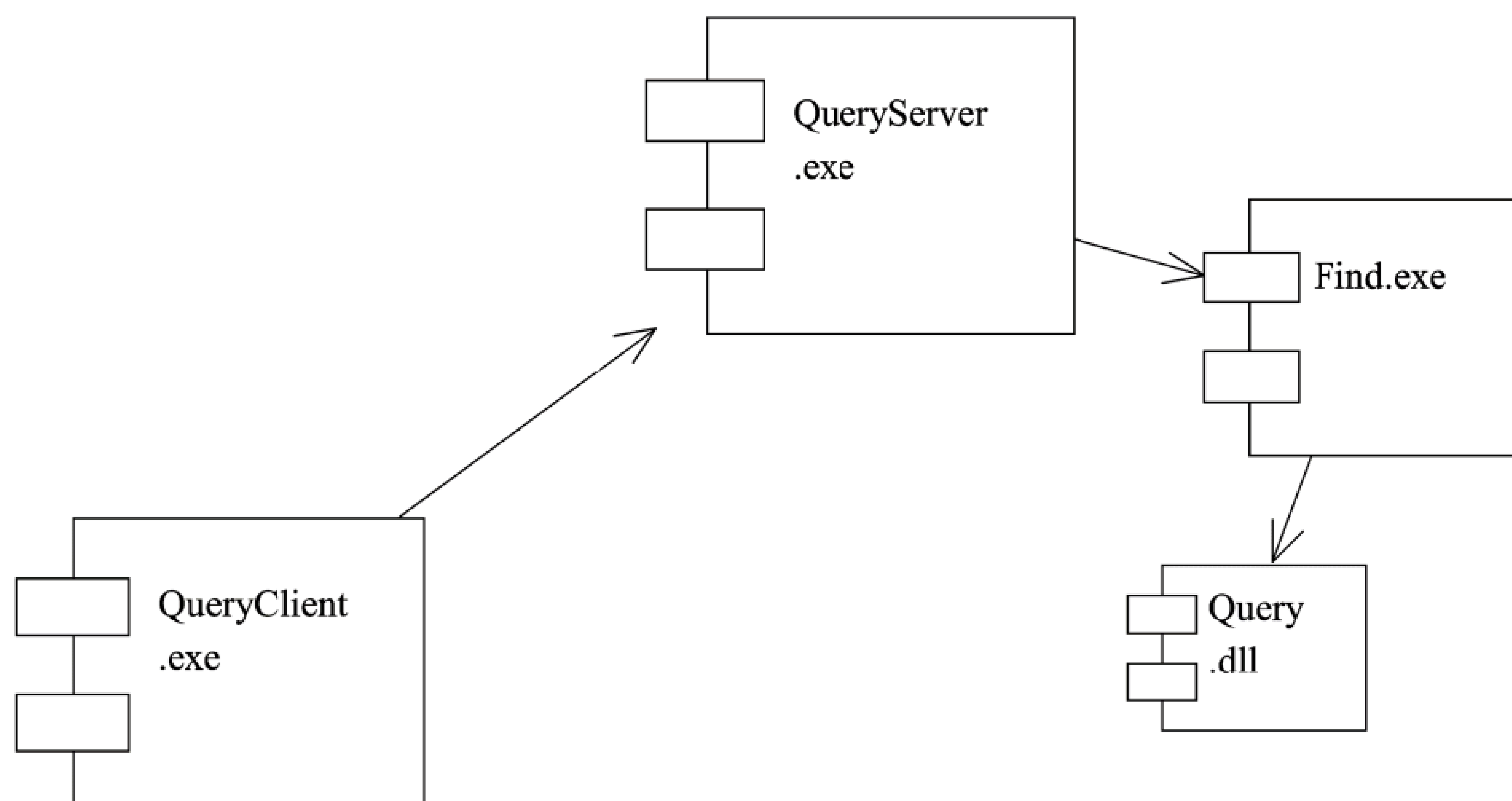


图 13-10 构件图示例

通常来说，可以使用构件图完成以下工作。

（1）对源代码进行建模：构件图可以清晰地表示出各个不同源程序文件之间的关系。

（2）对可执行体的发布建模：如图 13-10 所示，将清晰地表示出各个可执行文件、动态链接库（Dynamic Link Library，DLL）文件之间的关系。

（3）对物理数据库建模：用来表示各种类型的数据库、表之间的关系。

（4）对可调整的系统建模：例如，对于应用了负载均衡、故障恢复等系统的建模。

在绘制构件图时，应该注意侧重于描述系统的静态实现视图的一个方面，图形不要过于简化，应该为构件图取一个直观的名称，在绘制时避免产生线的交叉。

13.9 部署图

部署图也称为实施图，构件图说明构件之间的逻辑关系，而部署图则是在此基础上更进一步，描述系统硬件的物理拓扑结构以及在此结构上执行的软件。部署图可以显示计算节点的拓扑结构和通信路径、节点上运行的软件构件，常常用于帮助理解分布式系统。

图 13-11 就是与图 13-10 对应的部署图，这样的图示可以使系统的安装、部署更为简单。在部署图中，通常包括以下一些关键的组成部分：

（1）节点（Node）和连接。节点代表一个物理设备以及其上运行的软件系统，如一

台 WINNT 主机、一个 PC 终端、一台打印机、一个传感器等。如图 13-11 所示，“客户端：个人 PC”和“服务器”就是两个节点。在 UML 中，使用一个立方体表示一个节点，节点名放在左上角。节点之间的连线表示系统之间进行交互的通信路径，在 UML 中被称为连接。通信类型则放在连接旁边的“《》”之间，表示所用的通信协议或网络类型。

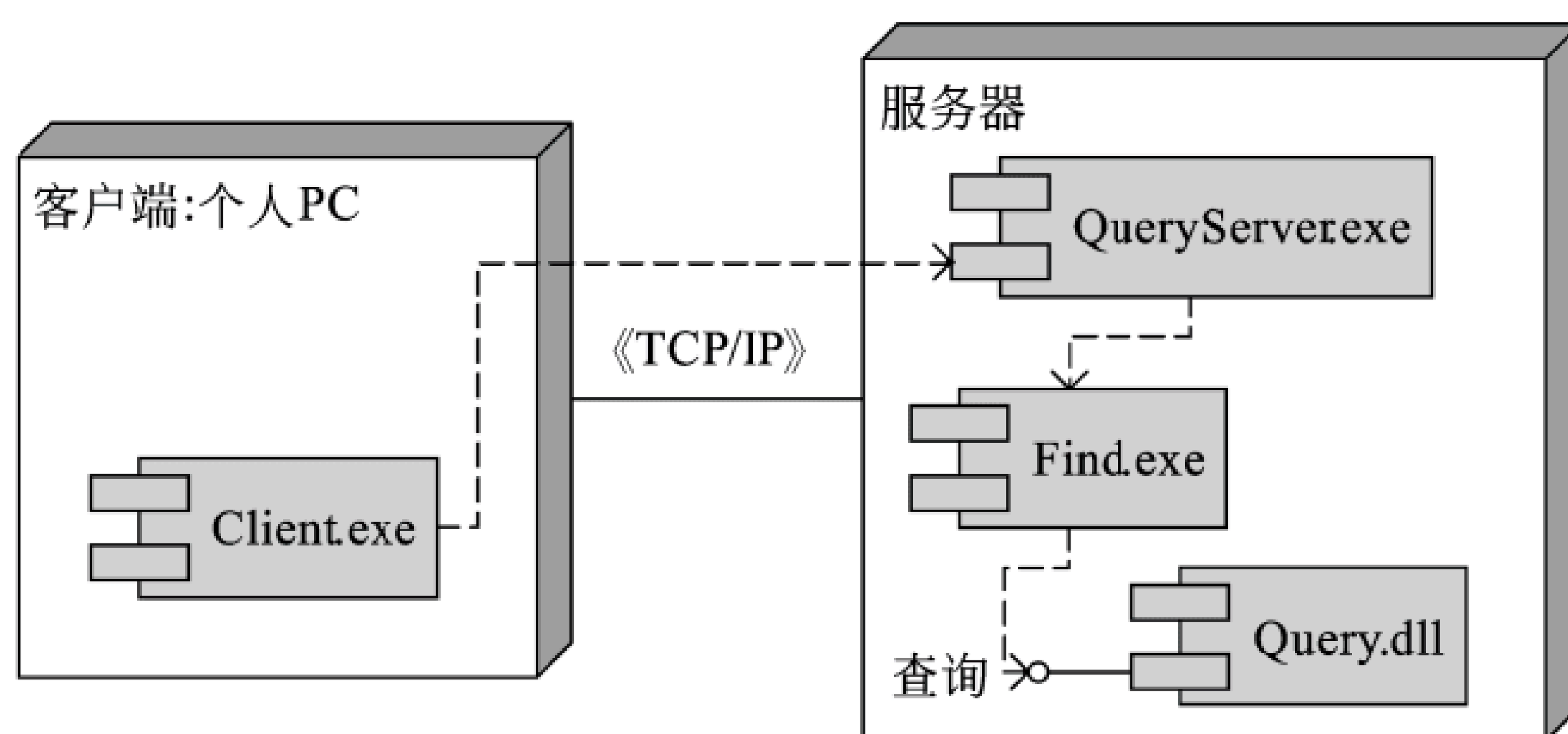


图 13-11　部署图示例

（2）构件和接口。在部署图中，构件代表可执行的物理代码模块，例如，一个可执行程序等。逻辑上它可以与类图中的包或类对应。例如，在图 13-11 中，“服务器”节点中包含 QueryServer.exe、Find.exe 和 Query.dll 三个构件。在面向对象方法中，类和构件等元素并不是所有的属性和操作都对外可见。它们对外提供了可见操作和属性，称之为类和构件的接口。界面可以表示为一头是小圆圈的直线。在图 13-11 中，Query.dll 构件提供了一个“查询”接口。

本章参考文献

[1] 刘超，张莉. 可视化面向对象建模技术. 北京：北京航空航天大学出版社，1999

[2] Wendy Boggs，Michael Boggs. UML 与 Rational Rose 2002 从入门到精通. 北京：电子工业出版社，2002

[3] 方贵宾，李侃，张罡. UML 和统一过程：实用面向对象的分析和设计. 北京：机械工业出版社，2003

[4] 黄晓春. UML：Java 程序员指南（双语版）. 北京：清华大学出版社，2004

[5] 徐家福. UML 精粹. 北京：清华大学出版社，2005

[6] 徐锋. AOSD 中文版—基于用例的面向方面软件开发. 北京：电子工业出版社，2005

第 14 章　统 一 过 程

统一过程（Unified Process，UP）是一个通用过程框架，可以用于种类广泛的软件系统的开发：不同的应用领域、不同的组织类型、不同的性能水平和不同的项目规模。UP 是基于构件的，这意味着利用它开发的软件系统是由构件构成的，构件之间通过定义良好的接口相互联系。在准备软件系统所有蓝图的时候，UP 使用的是 UML。

14.1　统一过程的特点

与其他软件过程相比，UP 具有三个显著的特点：用例驱动、以基本架构为中心、迭代和增量。

1．UP 是用例驱动的

开发软件系统的目的是要为该软件系统的用户服务。因此，要创建一个成功的软件系统，必须明白其潜在用户需要什么。

用例驱动是指开发过程将遵循一个流程：它将按照一系列由用例驱动的工作流程来进行。首先是定义用例，然后是设计用例，最后，用例是测试人员构建测试用例的来源。

尽管确实是用例在驱动整个开发过程，但是并不能孤立地选择用例。它们必须与系统架构协同开发。也就是说，用例驱动系统架构，而系统架构反过来又影响用例的选择。因此，随着生命期的继续，系统架构和用例都逐渐成熟。

2．UP 是以架构为中心的

架构根据企业的需求来设计，而这种需求则是由用户和其他项目干系人所感知，并反映在用例之中。然而，它还受其他许多因素的影响，例如，软件运行的平台（如计算机基本结构、操作系统、数据库管理系统和网络通信协议等）、可得到的可再用构件（如图形用户界面框架等）、配置方面的考虑、已有系统和非功能性需求（如性能和可靠性等）等。

架构设计师将软件系统构筑在一个框架中，正是这个框架必须被设计成让系统不仅在初始开发期间，而且在未来的版本演化过程中能不断发展。要找到这样的一个框架，架构设计师必须对系统的关键功能也就是系统的关键用例有一个总体性把握。这些关键用例虽然只占用例总数的 5%～10%，但是，它们却是最重要的，因为它们将构成整个系统的核心功能。下面是这个过程的简单描述：

（1）首先，架构设计师从不与特定的用例相关的部分（如平台）着手来创建基本架

构的大致轮廓。尽管基本架构的这部分是与用例无关的，但是，在建立基本架构的轮廓之前，架构设计师必须对用例有一个总体性把握。

（2）其次，设计人员应当从已经确认的用例子集着手开始工作，这些用例是指那些代表待开发系统的关键功能的用例。每个选定的用例都应当被详细描述，并在子系统、类和构件层次上实现。

（3）随着用例已经被定义并且逐渐成熟，基本架构就越来越成形了。而这种状况，反过来又导致更多用例的成熟。

这个过程会不断持续下去，直到基本架构稳定为止。

3．UP 是迭代式的和增量的

UP 开发一个商业软件产品是一项可能持续几个月、一年甚至更长时间的工作。因此，将此种工作分解成若干更小的部分或若干小项目是切合实际的。每个小项目都是能导致一个增量的一次迭代过程。迭代指的是工作流中的步骤，而增量指的是产品的成长。

开发人员根据两个因素来选择在一次迭代中要实现什么。首先，迭代与一组用例相关，这些用例共同扩展了到目前为止所开发的产品的可用性。其次，迭代涉及最为重要的风险。后续迭代是建立在先前的迭代完成后的开发成果之上的。在每次迭代中，开发人员识别并详细定义相关用例，利用已选定的基本架构作为指导来建立一个设计，以构件形式来实现该设计，并验证这些构件满足了用例。如果一次迭代达到了它的目标，那么开发过程就进入下一次迭代的开发了。当一次迭代没有满足它的目标时，开发人员必须重新审查先前的决定，尝试新的方法。

14.2　统一过程生命周期

UP 中的软件过程在时间上被分解为 4 个顺序的阶段，分别是初始阶段（inception）、细化阶段（elaboration）、构建阶段（construction）和交付阶段（transition）。每个阶段结束时都要安排一次技术评审，以确定这个阶段的目标是否已经满足。如果评审结果令人满意，就可以允许项目进入下一个阶段。基于 UP 的软件过程模型如图 14-1 所示。

从图 14-1 中可以看出，基于 UP 的软件过程是一个迭代过程。通过初始、细化、构建和提交 4 个阶段就是一个开发周期，每次经过这 4 个阶段就会产生一代软件。除非产品退役，否则通过重复同样的 4 个阶段，产品将演化为下一代产品，但每一次的侧重点都将放在不同的阶段上。这些随后的过程称为演化过程。

用户需求的变化、运行环境的变更、基础技术方面的变更等都会引发演化过程。通常情况下，演化过程的初始阶段和细化阶段都比较简单，因为基本产品定义和架构在前面的开发过程就已经决定。但也有例外情况，例如，对架构进行重新定义的演化过程。

在进度和工作量方面，所有阶段都各不相同。尽管不同的项目有很大的不同，但一个中等规模项目的典型初始开发周期应该预先考虑到工作量和进度间的分配，如表 14-1 所示。

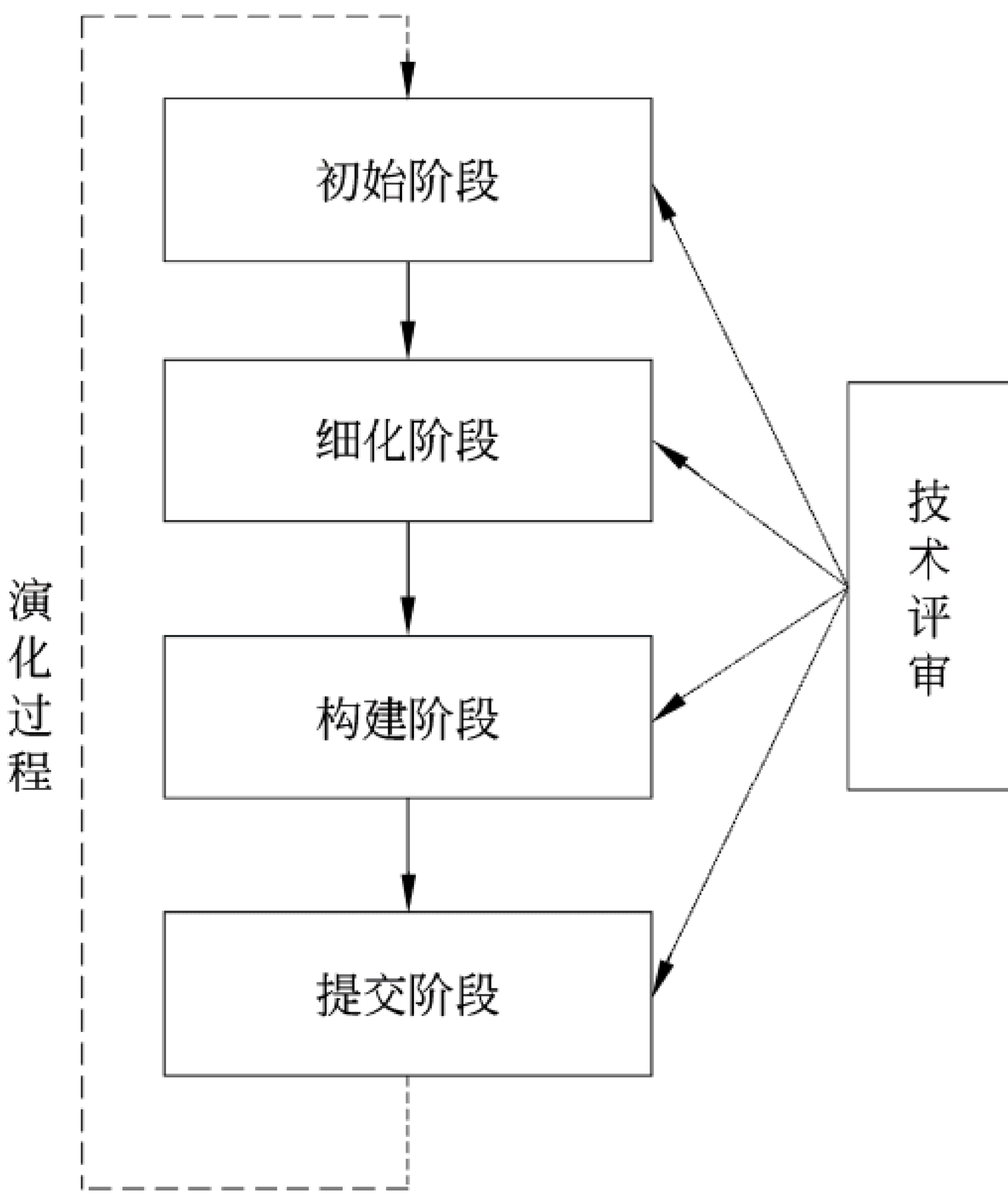


图 14-1　基于 UP 的软件过程

表 14-1　UP 各阶段的工作量和进度分配

	初 始 阶 段	细 化 阶 段	构 建 阶 段	提 交 阶 段
工作量	5%	20%	65%	10%
进度	10%	30%	50%	10%

对于演进周期，初始和细化阶段就小得多了。能够自动完成某些构建工作的工具将会缓解此现象，并使得构建阶段比初始阶段和细化阶段的总和还要小很多。

14.2.1　初始阶段

初始阶段的任务是为系统建立业务模型并确定项目的边界。在初始阶段，必须识别所有与系统交互的外部实体，定义系统与外部实体交互的特性。在这个阶段中所关注的是整个项目的业务和需求方面的主要风险。对于建立在原有系统基础上的开发项目来说，初始阶段可能很短。初始阶段的实现过程如图 14-2 所示。

1．明确项目规模

建立项目的软件规模和边界条件，包括验收标准、了解环境及重要的需求和约束、

识别系统的关键用例。

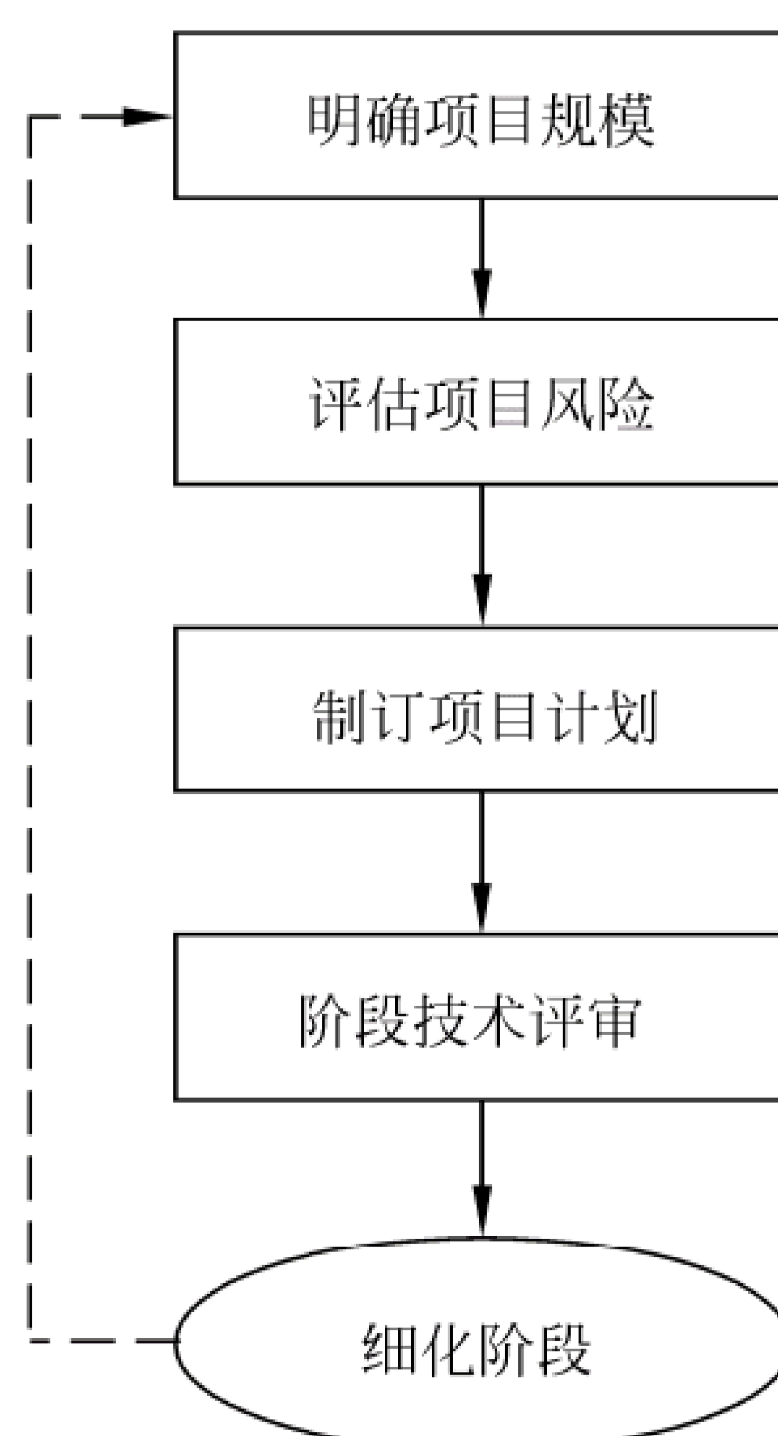


图 14-2　初始阶段子过程

2．评估项目风险

软件过程主要关心的是软件开发的已知方面，只能准确描述、计划、分配和评审那些已经知道将要完成的事情。风险管理则主要关心未知方面。在基于 UP 的迭代式软件开发过程中，很多决策要受风险决定。要达到这个目的，开发者需要详细了解项目所面临的风险，并对如何降低或处理风险有明确的策略。

3．制订项目计划

估计整个项目的总体成本、进度和人员配备。综合考虑备选架构，评估设计和自制/外购/重用方面的方案，从而估算出成本、进度和资源。在这个过程中，要通过对一些概念的证实来证明可行性，该证明可采用可模拟需求的模型形式或用于探索高风险区的初始原型。初始阶段的原型设计工作应该限制在确信解决方案可行就可以了，具体实现留到细化阶段和构建阶段。

4．阶段技术评审

初始阶段结束时要进行一次技术评审，检查初始阶段的目标是否完成，并决定继续进行项目还是取消项目。在评审过程中，需要考虑项目的规模定义、成本和进度估算是否适中，估算根据是否可靠？需求是否正确，开发方和用户方对软件需求的理解是否达成一致？是否已经确定所有风险，并且有针对每个风险的规避策略等问题。

14.2.2 细化阶段

细化阶段的任务是分析问题领域，建立健全的架构基础，淘汰项目中最高风险的元素。在细化阶段，必须在理解整个系统的基础上，对架构做出决策，包括其范围、主要功能和诸如性能等非功能需求，同时为项目建立支持环境。细化阶段的实现过程如图 14-3 所示。

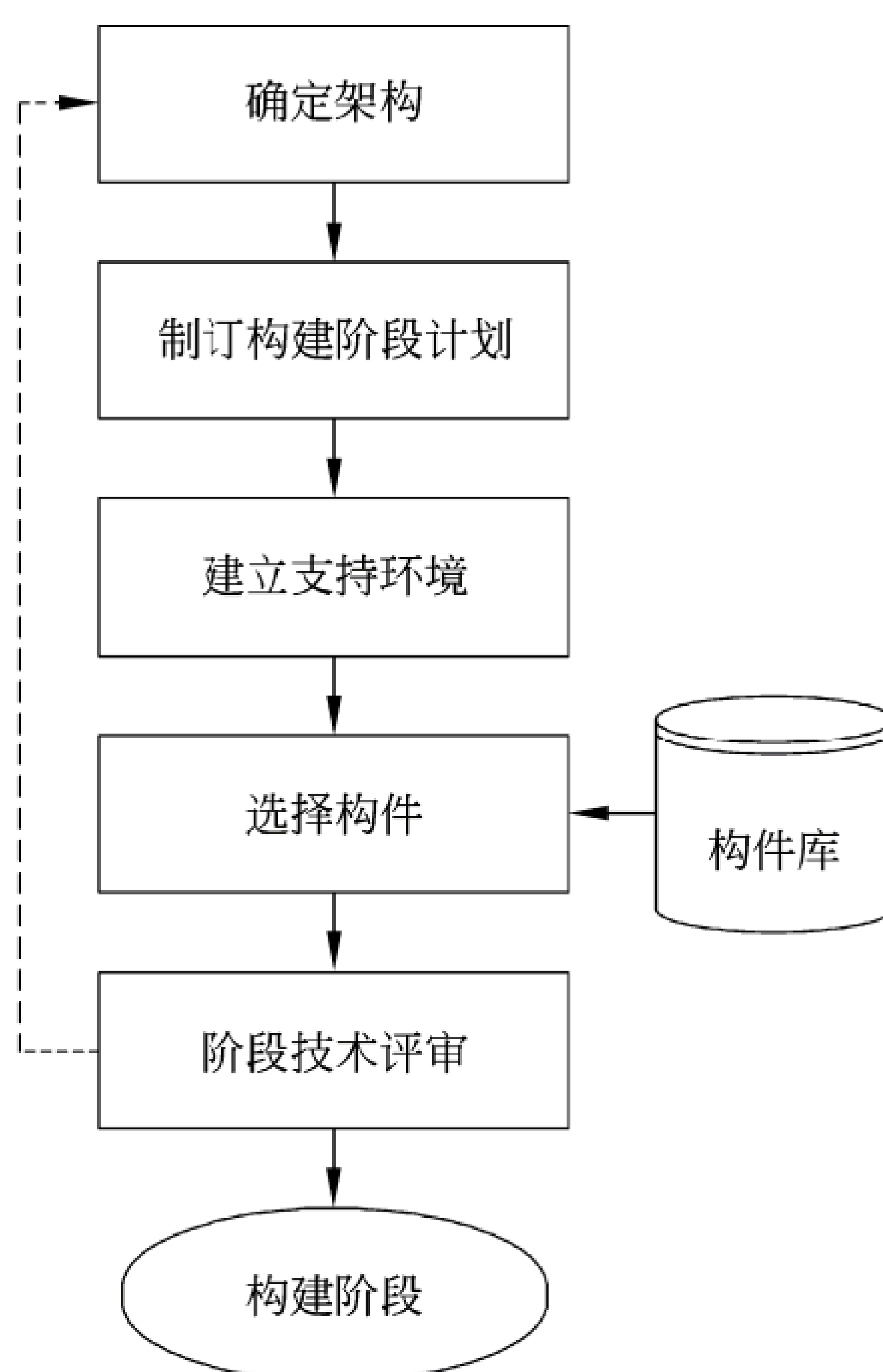


图 14-3 细化阶段子过程

1．确定架构

确保架构、需求和计划足够稳定，充分减少风险，从而能够有预见性地确定开发所需的成本和开发进度。通过处理架构方面重要的场景，建立一个已确定基线的架构。证明已建立基线的架构将在适当时间，以合理的成本支持系统需求。

2．制订构建阶段计划

为构建阶段制订详细的过程计划并为其建立基线。

3．建立支持环境

建立支持环境，包括开发环境、开发流程、支持构建团队所需的工具和自动化/半自动化支持。

4．选择构件

评估现有的构件库和潜在构件，充分了解自制/外购/重用决策，以便有把握地确定构建阶段的成本和进度。集成所选构件，并按主要场景进行评估。

5．阶段技术评审

评审时，需要检验详细的系统目标和范围、架构的选择以及主要风险的解决方案。在技术评审中，需要考虑以下一些问题。

（1）产品需求是否稳定，架构是否是稳定的？

（2）可执行原型是否表明已经找到了主要的风险元素，并且得到妥善解决？

（3）构建阶段的迭代计划是否足够详细和真实，是否有可靠的估算支持，可以保证工作继续进行？

（4）所有与项目有关的人员是否一致认为，如果在当前架构环境中执行当前计划来开发完整的系统，则当前的需求可以实现？

（5）实际的资源耗费与计划的耗费相比是否有偏差，该偏差是否可以接受？

14.2.3　构建阶段

在构建阶段，要开发所有剩余的构件和应用程序功能，把这些构件集成为产品，并进行详细测试。从某种意义上说，构建阶段是一个制造过程，其重点放在管理资源及控制操作，以优化成本、进度和质量。

构建阶段的主要任务是通过优化资源和避免不必要的报废和返工，使开发成本降到最低；完成所有所需功能的分析、开发和测试，快速完成可用的版本；确定软件、场地和用户是否已经为部署软件做好准备。

在构建阶段，开发团队的工作可以实现某种程度的并行。即使是较小的项目，也通常包括可以相互独立开发的构件，从而使各团队之间实现并行开发。这种并行性在较大幅度地加速开发进度的同时，也增加了资源管理和工作流程同步的复杂程度。

构建阶段结束时也要进行技术评审，评审产品是否可以在 β 测试环境中进行安装和运行。在评审中，需要考虑以下一些问题。

（1）该产品发布版是否足够稳定和成熟，可安装和运行在用户的实际环境中？

（2）所有与项目有关的人员是否已准备好将产品发布给用户？

（5）实际的资源耗费与计划的耗费相比是否有偏差，该偏差是否可以接受？

14.2.4　交付阶段

当基线已经足够完善，可以安装到最终用户实际环境中时，则进入交付阶段。交付阶段的重点是确保软件对最终用户是可用的。

交付阶段的主要任务是进行 β 测试，制作产品发布版本；对最终用户支持文档定稿；按用户的需求确认新系统；培训用户和维护人员；获得用户对当前版本的反馈，基于反

馈调整产品，如进行调试、性能或可用性的增强等。

根据产品的种类，交付阶段可能非常简单，也可能非常复杂。例如，发布现有桌面产品的新发布版可能十分简单，而替换一个国家的航空交通管制系统可能就非常复杂。

交付阶段结束时也要进行技术评审，评审目标是否实现，是否应该开始演化过程，用户对交付的产品是否满意等。

14.2.5 技术评审

在每个阶段结束时都要进行一次技术评审，以确定在完成该阶段的最终迭代后是否应该让项目进入下一阶段。技术评审要考虑的主要问题应该主要与项目管理有关，因为主要的技术问题应该已经在该阶段的最终迭代以及随后的活动中得到解决。技术评审的步骤如图 14-4 所示。

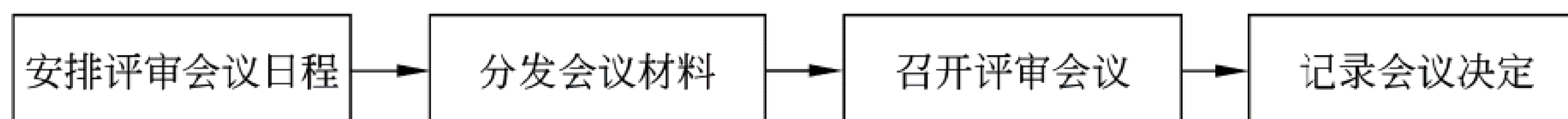


图 14-4 技术评审的步骤

1．安排评审会议日程

技术评审会议的参加者必须包括外部人员（用户代表和领域专家）、项目的管理团队（项目经理以及项目团队各功能区域的团队负责人）和项目评审委员会。

与会者一旦确定，就应安排会议的召开日期和时间，以便为与会者留出充足的准备时间，让他们能够评审有关材料。

2．分发会议材料

在会议召开之前，应当将技术评审材料分发给评审人员。要在会议召开之前及早地将这些材料分发出去，让评审人员有充足的时间对其进行审阅。

3．召开评审会议

在会议期间，评审人员主要关注状态评估。在会议结束时，评审人员应作出是否批准的决定。技术评审会议可能会得到以下结果之一。

（1）阶段被接受：评审委员会认为项目实现了该阶段的预期目标，可以进入下一阶段。

（2）有条件接受：评审委员会同意项目可以进入下一阶段，但必须先完成指定的纠正操作。如果发现的问题很少并且不是很重要，则客户可能决定在项目团队执行某些纠正操作的同时有条件地接受该产品。在这种情况下，项目经理需要根据问题的重要性，或选择开始新的迭代，以处理所出现的问题，或只是通过延长最终迭代来处理问题，二者的差异在于所需的计划工作量。

（3）阶段不被接受：项目没有实现该阶段的预期目标，项目经理就可能必须开始另一次迭代，甚至项目经理无法决定对问题的解决方案，而需要由有关人员根据合同重新

确定项目规模或终止项目。

4．记录会议决定

在会议结束时应完成评审记录，其中包括重要的讨论或活动以及评审的结果。如果结果是“阶段不被接受”，则应暂时安排一次后续复审。

14.3　统一过程项目管理

UP 的工作流程分为两部分，分别是核心工作流程和核心支持工作流程。核心工作流程（在项目中的流程）包括业务需求建模、分析与设计、实现、测试、部署；核心支持工作流程（在组织中的流程）包括环境、项目管理、配置与变更管理。

1．业务需求建模

业务需求建模的目的在于了解目标组织（将要在其中部署系统的组织）的结构及机制，以及目标组织中当前存在的问题，并确定改进的可能性，确保客户、最终用户和开发人员就目标组织达成共识，导出支持目标组织所需的系统需求。

为实现这些目标，业务建模工作流程说明了如何拟定新目标组织的前景，并基于该前景来确定组织在业务用例模型和业务对象模型中的流程、角色以及职责。作为对这些模型的补充，还需要编写补充业务规约和词汇表。业务建模工作流程如图 14-5 所示。

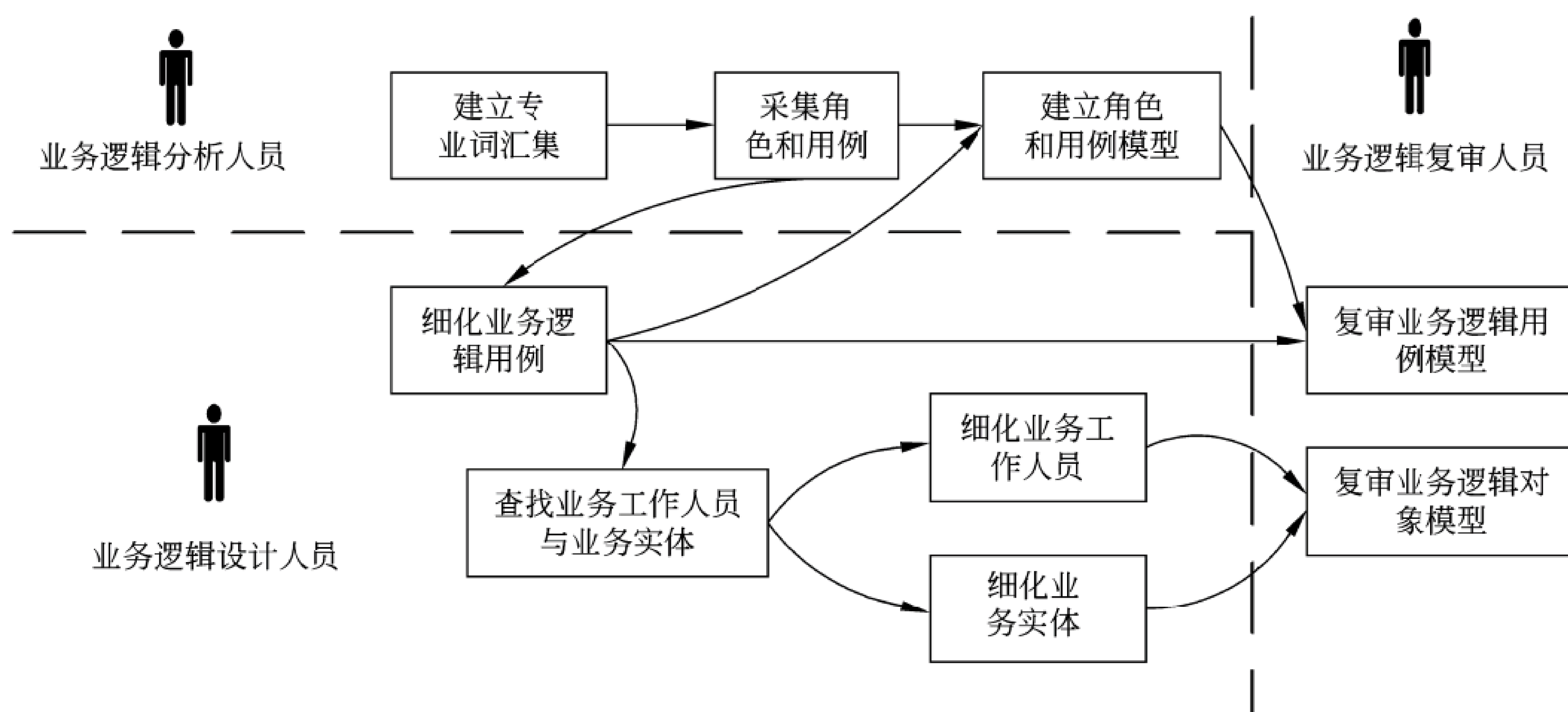


图 14-5　业务需求建模工作流程

业务需求建模工作所提供的文档与模型有业务逻辑建模（Rose）、业务需求说明书（MS Word）、专业词汇表（英汉对照）（MS Word）、风险说明（MS Word）、复审说明书（MS Word）。

2．分析与设计

分析与设计的目的在于将业务需求转换为未来系统的设计，逐步开发强壮的系统架构，使设计适合于实施环境，为提高性能而进行设计。分析与设计工作流程如图 14-6 所示。

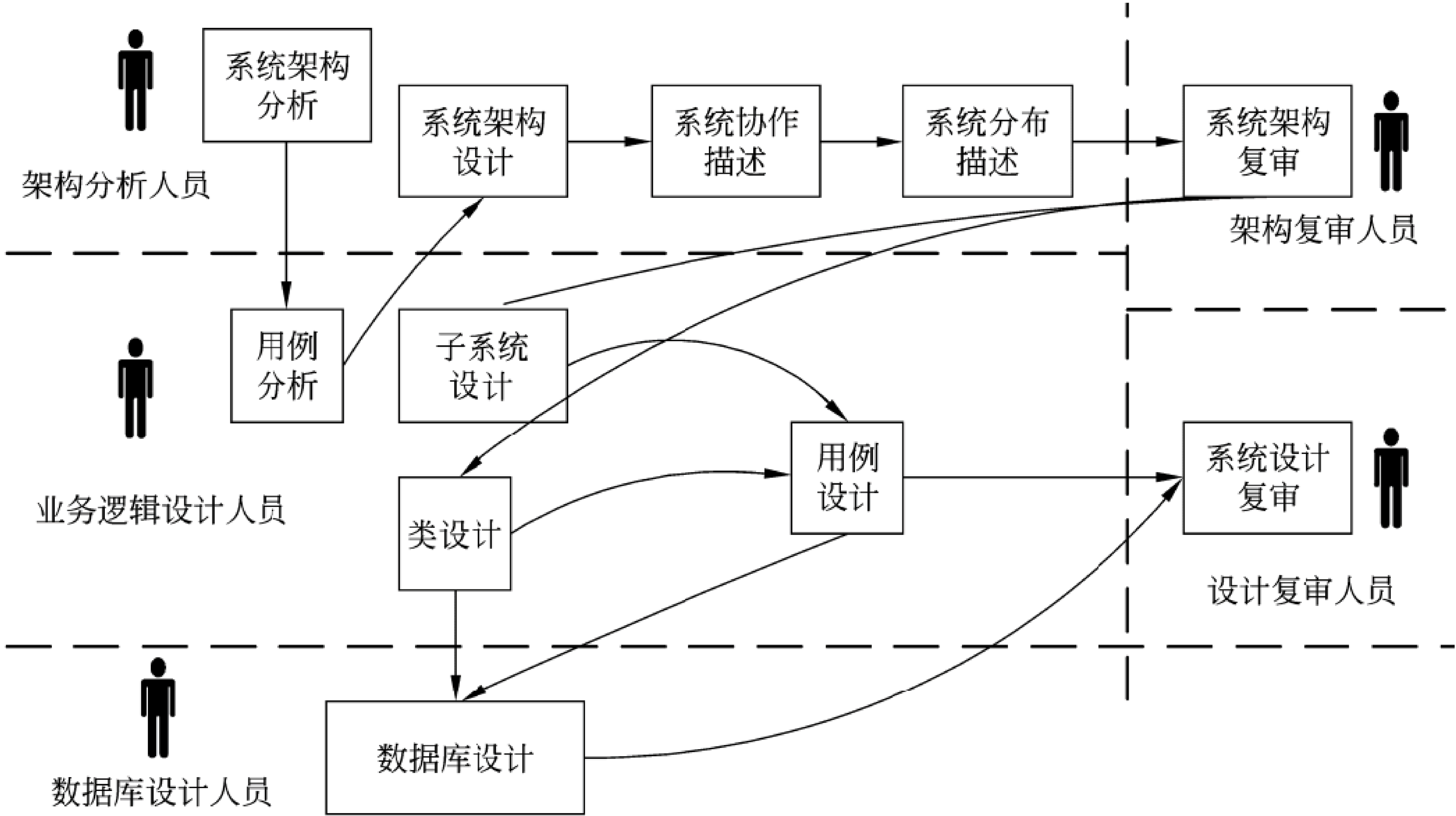


图 14-6　分析与设计工作流程

分析与设计工作所提供的文档与模型有系统总体设计报告（Word）、系统设计模型（Rose）、系统设计模型（Rose）、数据库设计模型（Power Designer）、数据字典（Word）、系统详细设计报告（Word）、工作量化说明书（Word）。

3．实现

实现的目的包括对照实现子系统的分层结构定义代码结构，以构件（源文件、二进制文件、可执行文件，以及其他文件等）的方式实现类和对象，对已开发的构件按单元来测试，并且将各开发人员（或团队）完成的结果集成到可执行系统中。

实现工作流程的范围仅限于如何对各个类进行单元测试，系统测试和集成测试将在测试工作流程中进行说明。测试的目的在于验证对象之间的交互、软件的所有构件是否正确集成、所有需求是否已经正确实现、确定缺陷并确保在部署软件之前将缺陷解决。

实现工作流程如图 14-7 所示。

实现工作所提供的文档与模型实现总结报告（MS Word）、实现模型（Rose）、系统集成说明书（MS Word）、代码审核意见报告（MS Word）、源代码（MS Word）、用户使用手册（MS Word）、错误解决记录手册（MS Word）、构件及其说明（MS Word）。

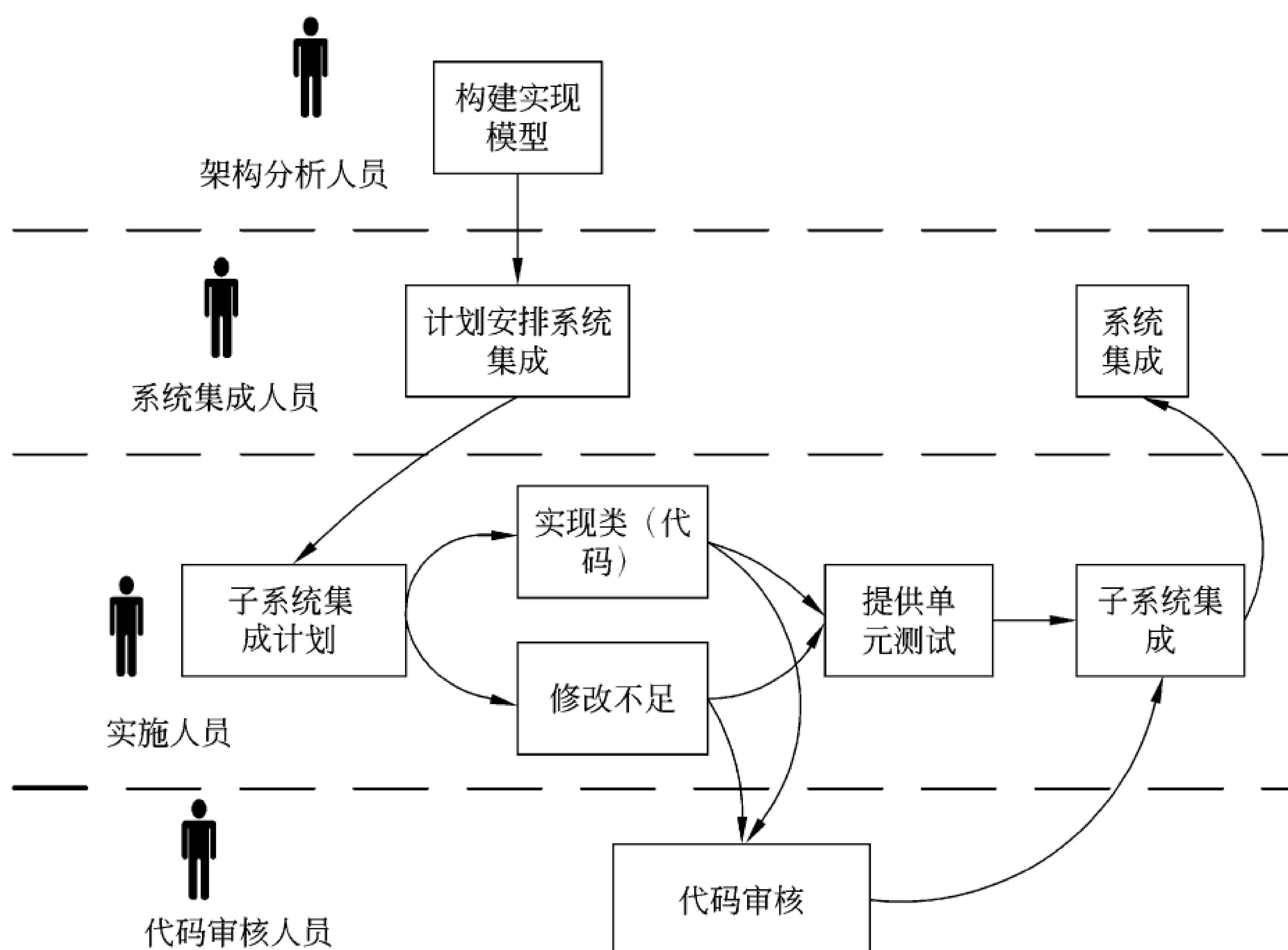


图 14-7　实现工作流程

4．项目管理

项目管理的目标是通过提供一些项目管理的环境，使这个任务更加容易完成。项目管理的目的是为对软件密集型项目进行管理提供框架，为项目的计划、人员配备、执行和监测提供实用的准则，为管理风险提供框架。该工作流程主要侧重于迭代式开发流程的以下重要方面：风险管理；计划迭代式项目，贯穿生命周期并针对特定的迭代；监测迭代式项目的进度、指标。

项目管理工作流程如图 14-8 所示。

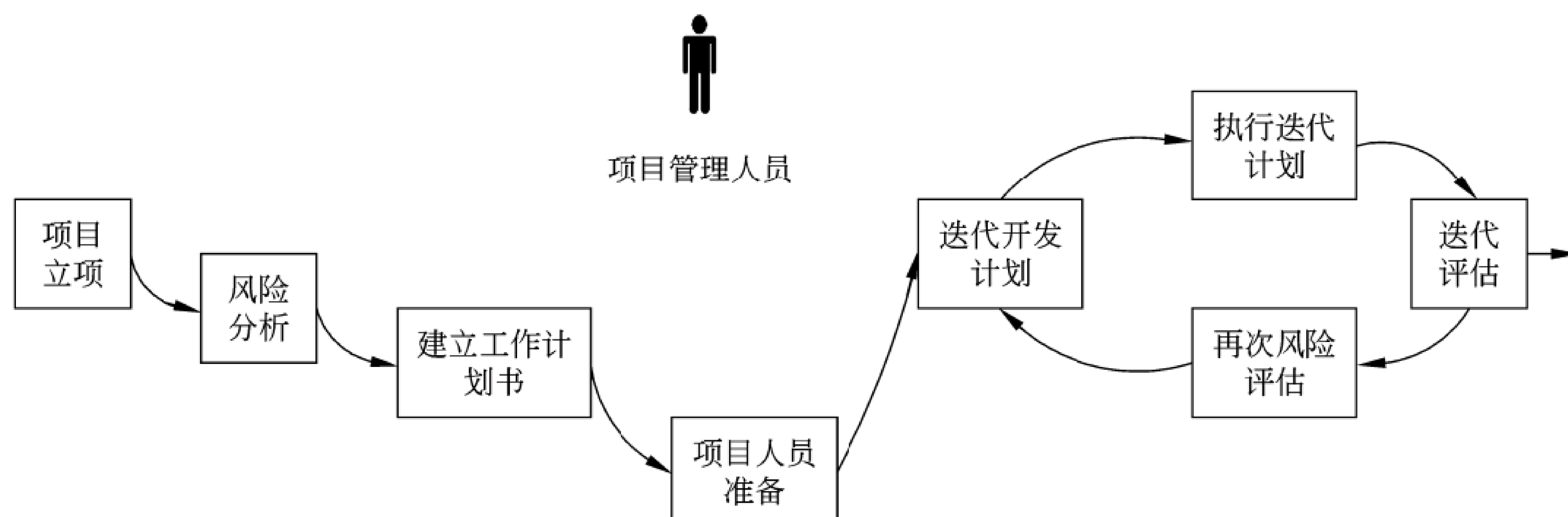


图 14-8　项目管理工作流程

项目管理工作所提供的文档和模型有风险管理计划（MS Excel）、工作计划书（MS Excel）、风险列表（MS Excel）、迭代计划（MS Excel）、问题解决计划（MS Excel）、测试计划（MS Excel）、系统集成计划（MS Excel）、子系统集成计划（MS Excel）、工作单（MS Excel）、产品验收计划（MS Excel）、评测计划（MS Excel）、项目计划复审意见（MS Word）、开发总结（MS Word）。

5．部署

部署工作流程用来描述那些为确保最终用户可以正常使用软件产品而进行的活动。

部署工作流程描述了两种产品部署的模式，分别为自定义安装和通过 Internet 使用软件。

在每个实例中，都强调要在开发场所对产品进行测试，并在产品最终发布之前进行 Beta 测试。尽管部署活动主要集中于产品化阶段，但在较早的一些阶段中也会有一些为部署进行计划和准备的活动。

部署工作所提供的文档和模板有部署计划（MS Word）、安装文档和发布说明（MS Word）。

本章参考文献

[1] Toshifumi T，Keishi S，ShinjiK. Improvement of Software Process by Process Description And Benefit Estimation. Proceeding of ICSE’95，Los Alamitos，CA，USA，IEEE，1995. 123-132

[2] Rational Software Corp. Rational Unified Process. 2000

[3] Jacobson，Booch L，Rumbaugh G. The Unified Software Development Process. Addsion Wcslcy，1999

[4] G. Rumbaugh，L. Booch，Jacobson. UML Reference Manual. Addison Wesley Longman，1999

[5] Jacobson，Griss M，Jonsson P. Software Reuse – Architecture，Process and Organization for Business Success，Addison Wesley Longman，1997

[6] 周之英，曾健. 一种软件过程模型. 清华大学学报（自然科学版），1998（S1）：57-60

[7] 曹会明，金茂忠，刘超. 一种有效的软件过程改善模型. 计算机工程与应用，2001（12）：89-92

[8] 刘军，王宁生. 基于UML的迭代式软件开发过程，计算机工程与应用，2001（17）：154-156

第 15 章　企业信息系统

香农认为：信息是系统有序程度的度量。诸如文字、数字、图画、声音、图像等，都是信息，香农给它们找到了统一的计量单位，那就是比特（bit）。换句话说，比特是信息的度量单位。

信息化（Informationalization）是一个日本学者在 1970 年提出的，至今还没有一个权威的定义。但这个词在中国应用的非常普遍。所谓信息化，可以被认为是现代信息技术与社会各个领域及其各个层面相互作用的动态过程及其结果。注意，信息化这个概念，既关注过程也关注结果。以笔者之见，信息化是一个无止境的工程，就像人类对自然世界的改造一样，随着信息技术的进步，随着人们对社会的认识，会不断地提高人们对信息资源的开发和应用水平。即便谈及信息化的结果，那也只能是阶段性的结果。

企业信息化，简单讲就是信息技术被应用到企业的过程。具体地讲，企业信息化包含管理信息化、生产过程信息化、设计信息化、制造工艺信息化，还有办公自动化等内容。随着计算机技术的进步，相关学科（包括管理科学）的进步，信息系统在企业的有效应用，可以减少时间的浪费，能量的浪费和物质的浪费。

15.1　企业资源计划

ERP 由美国著名的计算机技术咨询和评估集团提出的一整套企业管理软件系统标准，遵循这一标准体系，全世界各管理软件公司在经历众多企业管理系统开发和应用的过程中，汇集不同行业不同企业的管理需求特点、管理模式和管理经验，不断完善和发展自己的 ERP 系统应用产品，形成了百花齐放，百家争鸣的市场格局。

ERP 的具体定义在业内有过许多说法。ERP 是一套多方面、全方位为企业运作提供辅助决策信息和大量日常管理信息的大规模集成化软件，同时也是企业管理不断向零缺陷趋近的一整套现代化管理思想和办公手段，它包括财务管理、供应链管理、客户关系管理、项目管理、人力资源管理、资产设备维护等诸多方面。它能使企业在纵横市场的过程中始终处于企业供应与市场需求的平衡点，以及最优资源匹配，最少资源占用的状态，从而加速企业资金周转，修正企业日常运作中的偏差，使企业达到全面受控状态。

15.1.1　ERP 的作用

ERP 的作用是在协调与整合企业各方面资源运作的过程中，全面实现信息共享和企业对市场变化的快速反应，降低市场波动给企业带来的经营风险，帮助企业以更少的资

源投入获得更多的投资回报。具体由如下几个方面来体现。

（1）在供应制造方面：通过物料清单（Bill of Material，BoM）、主生产计划（Master Production Schedule，MPS）等管理功能，帮助企业达到“以销定产，以产定料，以料的需求来花钱”这一良性循环。从而降低企业资金在供应仓库、生产车间、产成品库等方面的固化，加速资金周转。

（2）在分销渠道方面：通过对订单、发货、信用、应收、调拨等相结合的管理措施，帮助企业规避或减少由于“牛鞭效应”带来的各种不良后果——最终消费需求波动在分销渠道链上的放大，导致供货不平衡带来的重复运输、仓库积压、商品短缺等现象而增加经营成本，以及由于经销商、代理商管理不善带来的欠款问题、串货现象、价格失控等管理漏洞。

（3）在集中财务管理方面：使财务管理水平从简单的会计核算向管理会计方面提高，例如，成本分析与控制、多级责任中心、多维核算与分析的记账基础、与业务密切结合的预算管理控制体系、审计追溯、财务报表合并、财务与业务的无缝集成、现金流管理与预测等。

（4）在客户关系管理方面：通过对销售过程的严密监控以及机会信息的分析，提高销售预测水平和业务人员的销售业绩；降低营销人员流失导致客户资源流失的损失；快速、低成本满足用户服务需求以及不断挖掘新、老客户的潜在价值。

（5）在人力资源管理方面：整合企业中的员工、人事、薪资福利、考勤等信息，有效规划人员职业生涯，推动企业人员管理从简单的劳资关系管理迈向人力资本化管理。

（6）在项目管理方面：通过高效率的信息化平台，快速收集、反映与分析各种项目资源的占用、空闲，并进行有效分配，同时将合同管理与项目任务、项目施工单位、项目经理密切结合，降低项目运作过程中的各种风险，缩短项目周期和运营成本。

（7）在资产维护方面：通过对生产设备及其相关零部件等的维修信息、运行信息、寿命信息的记录、追踪与分析，指导设备部件的准确采购和及时修复，降低企业特别是连续流程式工厂的大量设备用备品、备件库存资金，防止和减少因设备隐患与故障造成工厂停产的重大损失。

15.1.2 ERP的发展过程

管理信息系统的发展大致经历了 MIS、MRP、MRPII、ERP 几个阶段，今天对 ERP 的定义是从广义 ERP 的角度来阐述的。而常见的 ERP 系统主要包括供应、制造、财务、分销、库存几个方面，随着 ERP 厂商不断扩充更多的功能，软件系统又囊括了高级计划系统（Advanced Planning System）、CRM、电子商务等方面，后来被称为 ERPII。

在早期的 MIS 系统阶段，软件开发主要从企业现有的业务处理模式出发，记录、计算大量原始数据，并支持查询、汇总等方面的工作。以计算机信息化管理来替代手工操作，但是存在的主要问题是：由于计算机模拟原有的手工操作方式，管理效率、效果依

然不尽人意，许多投入付之东流，信息化的投入带来的产出很少，有时甚至不能覆盖高昂的系统维护费用，这种现象被称为“IT 黑洞”。

由于制造企业，特别是机械加工装配企业产品品种繁多，零部件多，数据处理量大，生产计划经常变更，且生产周期长，材料、零部件计算量大，特别是在生产过程中，常常出现缺料、缺件的现象，往往一些不起眼的零部件的缺失，造成最终产品无法装配、影响准时交货，给企业带来经营风险和重大损失。企业迫切需要按照产品组成结构对应的零部件耗用关系、按照生产工艺中规定的加工周期进行材料需量、缺量计算，以及采购、制造提前期计算的软件系统。MRP 系统便应运而生，MRP 系统的核心模块是 BOM，系统可对产品构成进行管理，借助计算机的运算能力及系统对客户订单，在库物料、产品构成的管理能力，实现依据客户订单，按照产品结构清单展开并计算物料需求计划，从而减少库存积压，优化库存结构的管理目标。

当企业业务繁荣的时候，常常发现产能不够，这时需要优先考虑生产那些重要客户的订货、利润较高以及交货期临近的产品。于是在 MRP 管理系统的基础上增加了对企业生产中心、生产能力等方面的管理，以进行生产能力分析和精确排产，并将财务的功能包括进来，在企业中形成以从采购、库存、制造到销售的闭环管理系统，并将财务的总账、应收、应付、成本、固定资产等与业务做到集成，这样系统已能动态监察到产、供、销的全部生产过程，并将这个过程资金化，从财务数据中体现出来，这就是 MRPⅡ系统。

进入 ERP 阶段后，企业管理从关注内部资源的协调，向供应商、客户方面进行延伸，并增加了供应商评估、客户信用控制、管理会计、质量管理和商业智能等功能，使企业管理更加精细化、集成化、智能化，从而成为一个真正成熟的管理平台。进入电子商务时代后，ERP 系统的支撑技术由 C/S 向基于 Internet 技术架构的 B/S 全面转型，并增加了电子交易市场、网上商店、供应商计划协同、网络采购平台等新的内容以适应市场变化对新型管理模式的需求。

15.2　供应链管理

著名经济学家道格拉斯•诺斯把生产的总成本划分为转型成本（Transformation Cost）和交易成本（Transaction Cost）。转型成本就是马克思主义经济学讲的物质变换成本，即人们通常所说的制造成本。交易成本包括了获取市场信息的成本、订立合同的成本、执行合同的成本等。理论经济学认为，降低转型成本的基本途径在于深化分工。然而，随着分工的深化，人们之间的相互依赖关系加深，交易关系愈益频繁，降低交易成本就成为一个具有决定意义的任务。

当今社会的经济活动中，几乎没有哪个产品不是通过社会协作来完成的。小到面包，大到汽车、飞机。从农民种植小麦开始，小麦收成后卖给面粉加工厂，面包加工企业在

流通环节买到加工好的面粉，完成面包的加工，加工好的面包再通过销售渠道的层层转移，摆上了超市的食品货架。面包的生产过程和销售过程还比较单一，汽车、飞机的制造过程将涉及到更多企业的协作，是更复杂的供需链（网）协作的结晶。越来越多的产业显示出对供应链的依赖。以汽车制造为例，今天的汽车制造已经越来越像 20 世纪 90 年代的计算机制造，对整车（机）厂而言，装配工序已经变得很简单，有竞争力的关键技术的研究越来越集中在几家核心的厂商，有竞争力的市场价格越来越依赖于对供应商的管理和对作业流程的管理，以及对质量的管理。

美国著名的物流专家马丁•克里斯托弗（Martin Christopher）认为，21 世纪的竞争将是供应链与供应链之间的竞争。

15.2.1 供应链的概念

供应链（Supply Chain）也称为供需链，它是以市场需求为起点（直接面对最终消费者，研究最终消费者究竟需要什么），以顾客（消费者）为中心，将顾客、供货商、供货商的供货商、制造商、分销商、零售商等环节的成员通过各种不同类型的合作方式，形成一条供应链，为顾客提供合适、合时、合价的产品。供应链的业务过程和操作，可以从工作流程（Work Flow）、实物流程（Physical Flow）、信息流程（Information Flow）和资金流程（funds flow）4 个方面进行分析。

（1）工作流程即交易与管理工作，是运用信息做出决定的过程。工作流程从消费者开始，最初的工作包括需求的分析、产品开发和设计、生产计划的制定，然后是商业和交易的发生，包括企业之间订立合同、承诺交易，还有执行方面的事宜，包括从安排生产到办理进出口文件再到落实销售的整个过程。

（2）实物流程是实物的交付和转移。包括运输过程、仓库的管理，以及包装分配等。

（3）资金流程是收取顾客货款和清偿供应商款项的过程。

（4）信息流程包含了信息的收集、处理和分析。信息流程不能简单地当作是一套计算机软件，在当今年代，只是计算机软件更方便作为信息的载体而已。信息流程实际是供应链上合作伙伴间的沟通活动。信息流程不是孤立存在的，它与供应链的工作流程、实物流程、资金流程共存，并把这三个流程有机地连接起来，成为一个整体。

供应链的信息流程带动工作流程，工作流程决定实物流程，实物流程反馈为资金流程。

从最初的原材料供应到最终的产品再到销售给最终消费者的整个过程叫做最终供应链（Ultimate Supply Chain）。复杂的最终供应链式有若干的段落供应链（Extended Supply Chain）组成，每段的段落供应链提供不同的部件或服务，如汽车制造的供应链一定是由若干的段落供应链所组成的。就每个企业个体来讲，每个企业也都是一个基本的供应链（Basic Supply Chain）。

当今的企业竞争已经演变为一个供应链与另一个供应链之间的竞争。某一产业内，

只能容纳 3～5 家甚至 2～3 家“通才型”企业，以其为核心形成该产业内的一条供应链，这些企业就成为链主。其余成千上万家企业只能是“专家型”企业，通过融入某一个供应链中求得发展，他们被称为节点企业。链主企业在供应链上起着“系统集成”的作用，做供应链的整合、管理和协同，也主导分配供应链上的收益。作为链主的企业通常都是强势品牌的拥有者，他们可以是品牌经营性的（如耐克等），可以是研发制造性的（如联想、希赛等），可以是流通性的（如沃尔玛、香港利丰等）。作为节点的企业只能根据自己拥有的核心能力和关键资源，找准自己在价值链中的定位，专注于价值链上的某一环节、某一区段。只有其在核心业务上的出色表现，链上的其他企业才愿意与之结为联盟。

松散的合作，合作各方缺乏信任，在利益上是“敌对”的关系，彼此压榨，争取自身的最大价值，这样做局部上是优化的，整体上未必最佳，加大了供应链上的沟通成本，降低了沟通效率。每一单的谈判，每一批次的质检都需要投入大量的工作，耗费大量的非增值资源。形成稳定的合作联盟，利益持续相关，信任大大加强，减少了许多重复的谈判，减少了很多供应链上重复的非增值流程。这样做可以达到合作伙伴分享利润，实现多方获利。

15.2.2　供应链管理的概念

SCM 是把生产过程从原材料和零部件采购、运输、加工、分销直到把产品送到最终消费者手中，作为一个环环相扣的完整链条，通过用现代信息技术武装起来的计划、控制和协调等经营活动，实现整个供应链的系统优化，和它各个环节的高效率的信息交换，达到成本最低，服务最好的目的。供应链管理是一套全新的管理理念，渗透到企业发展战略、生产策略、市场营销、财务管理、信息系统和人力资源等方面。供应链管理强调环节与环节之间的配合与效率，通过合作与优化流程去创造价值。

一个产品在市场上参与交易的最终价格由整个供应链上的成本来决定，有统计数据指出，现在市场上的工业产品中，其转型成本只占到总成本的 1/4 左右，其余 3/4 是被设计开发、市场营销、管理费用等吃掉了。以耐克为例，一双市场售价 100 美元的鞋子，其材料成本 15.67 美元，直接劳动成本 2.58 美元，管理费用 4.56 美元，工厂利润 1.9 美元。一目了然，依靠压缩制造费用和控制材料成本对最终产品的竞争力的影响已经不大。需要对供应链的流程作重新的组合和优化，对供应链上的资源作重新分配，从而提高供应链的交易效率，降低供应链的交易成本，增强供应链的快速反应能力，加速供应链上的资金流转。

所有企业都具备五项基本活动，采购、制造、运输、存储、销售，可以利用供应链管理的概念制订出相应的近期和远期计划。近期计划是针对某一种情况而制定，是具体操作的内容，远期计划则是设计整个企业的供应链方案。

近期供应链计划是将如何满足客户需求的答案具体化，使之成为行动方案，将供应链各个环节的工作定义妥当，确定各个环节参加的企业和单位，具体如一个生产、采购

或销售计划；远期供应链计划是确立供应链结构，形成对应的企业组织架构，并规定各个流程的操作模式。从战略层面做出部署，包括如何建立和维持供应链合作伙伴的合作关系及合作模式、发展方向等。表 15-1 是一个供应链近期计划与远期计划的例子。

表 15-1　供应链近期计划与远期计划

活　　动	近 期 计 划	远 期 计 划
采购	购买的具体细节，数量，到货时间	供货商的选择，与供货商长期合作的内容
制造	生产线的安排，具体操作流程，支援和人员调配	厂址的选择，产能储备，各地生产配置
运输	路线、货运模式的安排	运输网络的配置，自营或外包
存储	配送的次序	存储设施所在地的确定
销售	供货的先后次序	未来的销售渠道，供应链的容量（能否应付促销活动）

SCM 离不开供应链上的企业实现信息共享，减少信息传递成本，减少信息传递的失真，减少信息沟通的障碍。信息系统是一个令每个环节都能获得决策和运作所需的信息，是工作快速完成的工具。依赖 SCM 系统，可以使具体操作自动化，资料更准确，处理更快速，并协助管理层和业务人员完成数据分析，制订和调整业务计划。有了信息系统的支撑，跟上市场的反应，抓住客户的需要，把需要促成为交易，实现供应链的收益。如果只有信息化的投入，没有客户需求的拉动，信息化的投入就完全成了运营的成本，对企业没有形成任何好处。

有专家把供应链上的合作总结为三种运行模式，分别是中心依附型、强强联合型和共生网络型。企业通过这三种合作模式可以应付不同的商业环境和平衡供需之间的相对优劣。

中心依附型是以一个企业为核心（链主企业），其他企业则是围绕在这个中心的独立单元，跟随这个中心企业来运作。这种模式里的链主企业，负责整个供应链的计划和调控，将这段供应链上的信息集中收集和发布。这种模式的供应链，通常各个供货商的谈判能力较弱，链主企业主宰着供应链的运行。一些巨大而技术密集的制造企业充当链主的角色，其以产品开发设计、销售和组装为核心能力，通过外包大量的零部件生产工序，控制整个供应链。

强强联合型是基于两家企业合作模式。两家企业由于在技术、市场、渠道等方面优势互补，但又不能单独掌握整条供应链的运作，所以组织成战略联盟，明确各自的分工，避免在对方领域里从事恶性竞争。

共生网络型是由很多相对独立的单元组成的相对松散、动态化的组织，各个单位通过每个订单、长期合作的经验而形成一个链网，互相依存，分工合作。共生网络的优势是，有很多中小型专门单位，可提供有弹性和专门的产品和服务，以专业的供应链管理

者作为中介或核心，利用非强制和柔性的管理，既能使整个供应链更灵活和敏捷，也能把各个单位组合起来，形成规模，优化整个供应链的配置。

希赛教育专家提示：SCM 不只是一套软件系统，而是一套因应市场挑战而形成的企业管理理论和企业运营方法。

15.2.3　供应链管理系统

供应链管理系统包括采购管理、库存管理、合同管理、销售管理、运输管理、生产订单、物料需求计划、主生产计划、粗加工能力计划、供应链计划、产能管理、设备维护、物料清单、产品配置、工艺流程、车间作业、委外加工、质量管理等。

1．采购管理

采购管理帮助采购人员控制并完成采购物料。涵盖从申请采购计划、采购下达至到货接收、检验入库的全部过程。可有效地监控采购计划的实施，采购成本的变动及供应商交货履约情况，从而帮助采购人员选择最佳的供应商和采购策略，确保采购工作高质量、高效率、低成本执行。采购需求信息可由生产等其他部门直接下达，无需手工录入采购订单，只要将申请采购项目合并下达即可自动生成采购单，方便、灵活。采购物品收货检验后可按已分配的库存货位自动入库，并及时更新库存，同时由成本与应付账子系统完成结转采购成本及应付款的工作，无须财务人员手工填制凭证。

2．库存管理

库存管理（Inventory Management）帮助企业的仓库管理人员对库存物品的入库、出库、移动和盘点等操作进行全面的控制和管理，达到提高库存控制精度，降低库存量，杜绝物料积压与短缺，提高客户服务水平，保证生产经营活动顺利进行的目的。库存管理子系统从归类、大类、货位、批次、单件、有效期等不同角度来管理库存物品的数量、货物流向、库存成本和资金占用情况，以便用户及时了解和控制库存业务各方面的准确情况。系统通常支持多种计量单位的自动转换，库存管理子系统为 ERP 系统的基础，所有的信息都由库存发出，并归结到财务上。

3．合同管理

合同管理（Contract Management）包括合同概要、收/付款计划、合同标的、合同条款、合同大事记、合同附件等信息处理，可以对合同进行生效/失效、变更、结案/弃结。应收/应付类合同可以填制合同结算单，确定执行数量、单价、金额等信息，通过“应收账管理”、“应付账管理”进行收付款和核销操作，收/付款信息回填合同结算单，并回写合同的执行数量、执行金额。用“资金管理”进行资金预测时，可以从“合同管理”中合同的“收/付款计划”采集数据，作为预测现金流动的一个依据。

4．销售管理

销售管理（Sales Management）帮助企业的销售人员完成客户档案管理、销售报价管理、销售订单管理、客户定金管理、客户信用检查、销售提货处理、销售发票处理、

客户退货及货款拒付处理和销售计划等一系列销售管理事务。销售管理子系统以订单为核心来管理整个销售业务。通过它用户可以及时了解到销售过程中每个环节的准确情况和数据信息。销售管理子系统作为企业运作中的一个重要部分，与库存、财务、生产等子系统有着紧密的联系。

5．运输管理

运输管理（Transportation Management）系统根据销售订单有关执行信息，进行运输计划（包括运输方式、运输路线、运输时间、运输货物等内容的制定）与预定（指与承运商、运输承包公司等的联系，运输工具型号、数量的指定与运输时间的预约），包装控制，装货操作、运输过程监控，交付处理等全流程控制，并进行各类装运文档（对于出口货物来说，各种必须的运输文档、包装文档显得特别重要）的管理。系统还可为分销合同、向供应商的索赔合同、项目订单（合同）及人力运输合同制定运输计划，并实施装运全流程操作。

6．生产订单

生产订单（Manufacture Order）是生产管理系统的最重要的组成部分。生产订单可以选择是否要做工艺流程管理。ERP 软件通常提供两种自动产生生产订单的方式：

（1）按照客户订单转生产订单。

（2）按照生产计划转生产订单，生产订单包括成品、半成品，以及外协类生产订单。

可选择按照物品标准 BOM 自动产生或由人工输入生产订单用材料清单，可选择按照物品标准 BOM 自动产生生产订单工艺流程表，或由人工输入生产订单工艺流程表，可由材料编号、工艺流程操作标准，自动产生生产订单别操作标准。

7．物料需求计划

MRP 是生产管理的核心，它的主要作用是将主生产计划安排的产品分解成各个自制零部件的加工装配计划和原料采购件的采购计划。同时，它和主生产计划、车间作业管理、库存管理等子系统形成了一个及时反映企业需要生产什么，什么时候生产，生产多少的动态闭环计划系统。因此物料计划编制的好坏，直接影响企业的效率，也反映了企业的管理水平。物料需求计划子系统能帮助企业摆脱过去按台套组织生产的管理方式，提供给企业一套全新的按零件组织生产的管理方式，从而提高企业的管理水平和经济效益。

按需求的来源不同，企业内部的物料可分为独立需求和相关需求两种类型。独立需求是指需求量和需求时间由企业外部的需求来决定，例如，客户订购的产品、科研试制需要的样品、售后维修需要的备品备件等；相关需求是指根据物料之间的结构组成关系由独立需求的物料所产生的需求，如半成品、零部件、原材料等的需求。MRP 的基本任务如下。

（1）从最终产品的生产计划（独立需求）导出相关物料（原材料、零部件等）的需求量和需求时间（相关需求）。

（2）根据物料的需求时间和生产（订货）周期来确定其开始生产（订货）的时间。

MRP 的基本内容是编制零件的生产计划和采购计划。要正确编制零件计划，首先必须落实产品的出产进度计划，用 MRP Ⅱ 的术语就是主生产计划（Master Production Schedule，MPS），这是 MRP 展开的依据。MRP 还需要知道产品的零件结构，即 BOM，才能把主生产计划展开成零件计划；同时，必须知道库存数量才能准确计算出零件的采购数量。因此，基本 MRP 的依据是 MPS、BOM、库存信息。

8．主生产计划

MPS 是确定每一具体的最终产品在每一具体时间段内生产数量的计划。这里的最终产品是指对于企业来说最终完成、要出厂的完成品，它要具体到产品的品种、型号。这里的具体时间段，通常是以周为单位，在有些情况下，也可以是日、旬、月。主生产计划详细规定生产什么、什么时段应该产出，它是独立需求计划。主生产计划根据客户合同和市场预测，把经营计划或生产大纲中的产品系列具体化，使之成为展开物料需求计划的主要依据，起到了从综合计划向具体计划过渡的承上启下作用。

9．粗加工能力计划

粗加工能力计划（Rough Cut Capacity Planning）针对主生产计划进行能力评估，主要用于评估关键资源能不能满足主生产计划的执行。评估现有的人员、设备、厂房、资金等资源能不能满足一个中长期计划的需要，以便于及时安排人员招聘、设备采购、资金调配等。

10．供应链计划

供应链计划（Supply Chain Planning）用于协调和整合供应链中各协作厂商的制造、采购计划。包括需求计划和风险管理、全球订单/产能/承诺管理、高级计划系统、跨供应链订单跟踪管理、供应链智能分析。

11．产能管理

产能管理（Capacity Management）用于协助企业有效地掌握车间产能负荷状况，并提供相关信息，以达下列目的：对已审核生产订单、已规划的生产订单，来规划生产订单在各工作中心的负载需求计算；分析各工作中心的产能/负载状况，是探讨长中短期生产计划可行性的依据。

12．设备维护

对于资产密集型，如：石化、钢铁等行业企业，特别是连续流程型企业，生产设备的投资大，管理复杂，生产设备的正常运转直接影响企业的经营。如何保障生产设备的正常运转，是稳定生产的重要前提。非计划的停机和不能在计划时间内完成维修活动都会影响产能的发挥，造成生产损失，降低产品质量，最终影响企业的利润和交付能力。

设备维护（Equipment Maintains）系统包括设备管理、检修管理、物资（备品备件）管理、技术文档管理等方面，通过基于周期、事件和状态参数等的预防性维护手段提高企业维护维修中的预见性和计划性，将企业员工尤其是生产人员从大量的事后处理的“救

火模式”下解放出来，更好地实现事前计划、事中控制、事后报告与分析等功能。

13．物料清单

MRP 系统要正确计算出物料需求的时间和数量，特别是相关需求物料的数量和时间，首先要使系统能够知道企业所制造的产品结构和所有要使用到的物料。产品结构列出构成成品或装配件的所有部件、构件、零件等的组成、装配关系和数量要求。它是 MRP 产品拆零的基础。当然，这并不是我们最终所要的 BOM。为了便于计算机识别，必须把产品结构图转换成规范的数据格式，这种用规范的数据格式来描述产品结构的文件就是物料清单。它必须说明构件（部件）中各种物料需求的数量和相互之间的组成结构关系。

14．产品配置

产品配置（Production Configuration）允许用户自由选择产品构件组成配套件，或自由设计产品，其具体做法为：事先为一个产品族分别建立不同的产品特征或为配套件定义多种构件可选项，在承接销售订单（合同）时根据客户需求选择产品的产品特征或配套构件，选择相应的生产路线和原材料，从而生成产品变体或配套件；同时，用户可模拟配置构件可用性与配套件的多角度成本核算。

15．工艺流程

工艺流程（Bill Of Routing）帮助企业对生产制造过程中的各个加工过程、操作标准进行准确而有效的管理，软件通常提供了三种产能与操作时间的管理方式，分别是按生产线、按工作单位、按设备群组。提供特殊日期、产品更换时间、非生产性时间的产能管理。

工艺流程管理系统为生产企业实施良好的质量管理与监控提供了高效的管理工具。它与 BOM 管理系统紧密结合在一起。构成了“生产管理系统”的两个重要的环节，支撑整个“生产管理系统”的有效运作。

16．车间作业

车间作业（Shop Floor Control）是企业短期计划管理的核心部分，车间作业管理主要是根据零部件的工艺路线来编制工序进度计划。车间作业管理子系统的核心部分就是根据 MRP 和工艺流程管理系统中提供的车间任务数据、产品配置数据、生产数据中提供的工艺路线数据、工作中心数据等编制车间进度计划。对需要下达的车间任务，首先进行模拟下达，检查物料、能力和工具的可用性。按照任务优先级数，分配物料和下达车间任务。按照工序优先级数，生成工作中心派工单。并将通过派工单实现从计划到实施的闭环控制，使车间管理人员了解车间任务完成情况。

车间作业系统的目标就是帮助用户监督和控制车间生产活动。帮助用户正确安排从车间进度计划生成、车间任务物料分配、车间任务下达、工作中心派工单生成到车间任务完工入库的全过程，以确保车间任务能够按照要求及时完成。同时帮助企业提高劳动生产率，减少车间在制品，提高产品质量。

17．委外加工

企业对采购进来的物料或自产的半成品，必须经过再加工处理后方可使用，而因某些原因无法进行自行加工处理，必须外包给其他合作厂商进行加工处理后再入厂使用。委外加工（Subcontracting）管理系统提供了对此进行管理的功能，同时对加工回厂的产品进行质量检验，以保证生产的顺利进行。

18．质量管理

通过对原材料质检信息、半成品质检信息、产成品质检信息以及产品售后的质量反馈信息的统计、分析，向企业的各级管理人员提供企业各个环节的质量分析报告，使他们了解质量信息及存在的问题，及时采取措施，避免不必要的损失，提高企业产品信誉。

质量管理（Quality Management）的好坏，直接关系着企业的命运，正所谓企业要靠质量求生存。

15.3　财务管理

现在，财务管理的解决方案已经完全融入到 ERP 中，成为 ERP 系统的一个核心部分。通过与业务系统的集成，实现财务管理与业务管理的一体化，实现资金流、物流、信息流的统一，增进了财务管理和领导决策的即时性、准确性。就财务软件本身的发展，经历了从核算型财务软件向管理型财务软件的发展，从单组织的财务电算化到多组织集团化企业的集中式集团财务管理。

15.3.1　财务管理软件的发展

企业在经营运作的过程中需要将人力、设备、材料、产品等一切资源资金化，只有将业务信息通过业务信息系统自动转化为相应的财务信息，自动生成凭证以及各种分析数据，并且根据财务制度自动对业务问题进行预警与监控，才能彻底杜绝企业管理中信息不能有效共享，重复录入、信息滞后、账账不符的现象，并规避各种管理风险。

从图 15-1 中，我们可以看到几个常见的财务模块（总账、固定资产、应收账、应付账、预算、现金流管理）与供应链模块（用圆角矩形表示）的对应关系。

这里举一个常见的例子，进一步分析财务业务集成的重要性。

对于一笔销售业务，在 ERP 系统中输入一条产品销售订单信息，可通过系统自动做凭证生成财务信息，如果赊销，则可生成应收款信息：在应收款管理模块生成一笔票据，在总账模块自动生成一笔凭证，借：应收账款，贷：销售收入和销项税，发货可结转销售成本、减少库存，可生成凭证，借：销售成本，贷：库存商品。这样，一个信息源（产品订货、发货信息）在系统中可自动转化成财务信息（应收票据、财务凭证）。

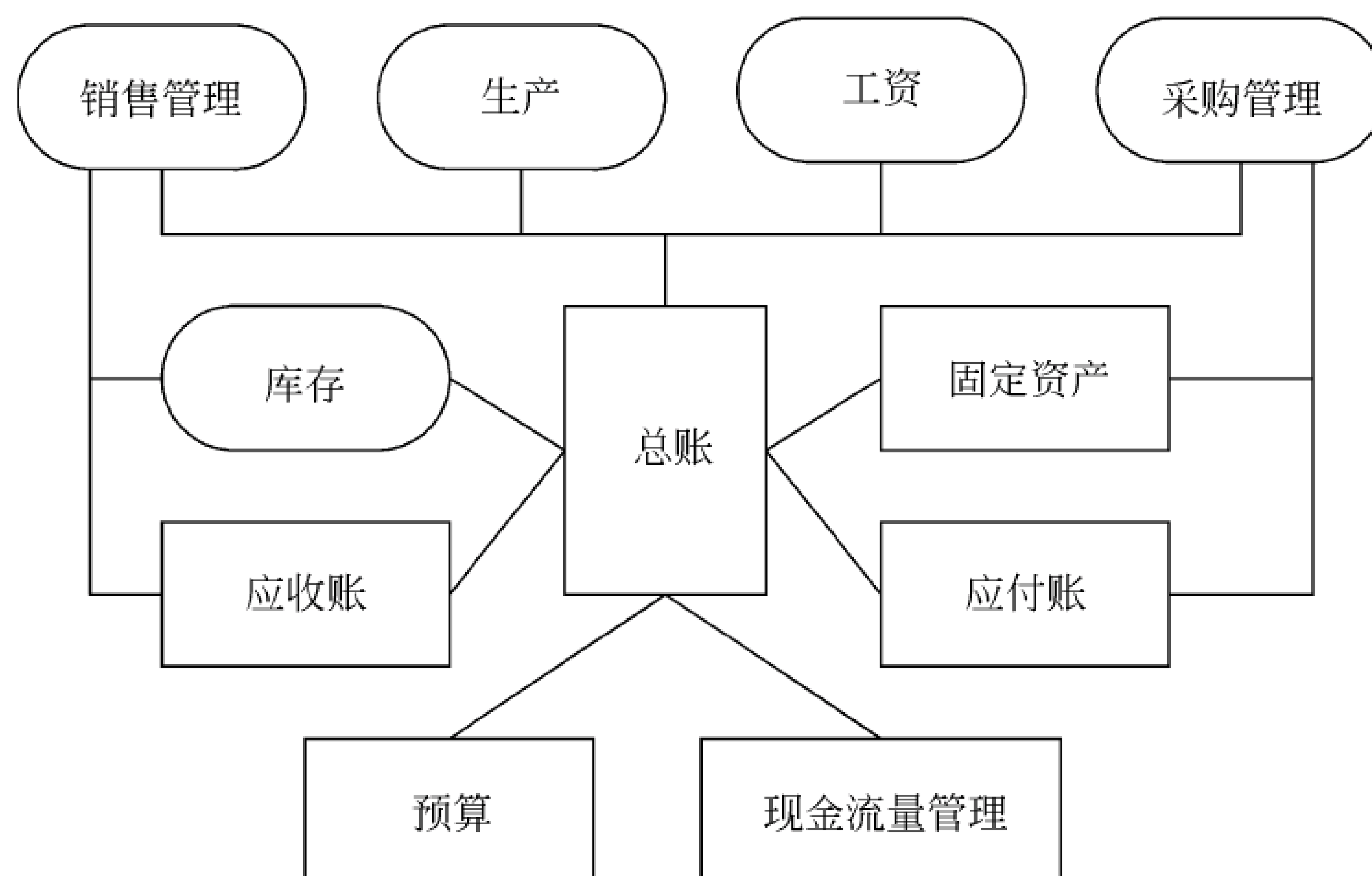


图 15-1 常见的财务模块与供应链模块的对应关系

那么，这样做能给企业带来哪些价值呢？这里有一个真实的案例。国内一家著名的制药企业，由于业务量大、变化频繁，在没有信息系统的时候，公司无法对应收账进行有效监控，欠款的详细信息都掌握在业务员手中，公司只知道哪个城市欠了多少钱，公司还设了一个纪检委书记专门追查销售业务中的腐败问题。

2005 年，这家企业实施了 ERP 系统，之后的一年里，公司销售增长 40%，而利润增长 50%，多出来的 10%的利润主要来自欠款的有效回收与控制。通过 ERP 系统，这家企业可以及时监控每一笔订单、每一次发货对应的欠款情况，以及客户回款是针对哪一笔业务。ERP 系统真正有效控制“应收账”，降低企业呆账、坏账损失，并保证销售回款。系统中有“授信额度”的概念，也就是说允许客户最大订单额是多少、最多可欠多少款、最大逾期未付款是多少。这些信息保存于每个客户的文件中，还可根据信用级别对客户分类。授信主管负责客户信用方面信息的管理和控制。可以设定“如果客户新的订单超过授信额度，将不能开出有关单据”，销售部门即使远在千里之外，也无法将产品卖给信用超额的客户。当然，对客户订单的归集、账龄分析、发催款单等功能都可对“应收账”进行有效的监控。过严的信用管理会影响销售额，过松的信用管理又容易产生坏账。有了 ERP 对“应收账”的综合管理，企业依据系统提供的信息，能有效平衡信用管理和销售额之间的关系，从而达到最大销售额与最小坏账损失。

由此，可以感受到使用集成化 ERP 系统之后，财务部门的作用发生了重大变化，不再只是单纯的记账，出报表，更重要的是从简单的会计核算和报送的职能向管理会计的职能转变。

还是上面的例子，有了集成化的信息，就可以非常方便快捷地进行审计追溯，从财务报表追溯到凭证，到客户的欠款情况，最终追溯到业务过程的业务单证。

企业在不断成长的过程中，随着跨地域、跨行业经营，分公司、子公司的设置，以

及各级责任中心的建立，财务体系也越来越复杂，对 ERP 系统的要求也越来越高。许多大型企业集团根据不同的产品线与业务单元设置了众多的事业部，各个事业部独立核算、独立运作，在业务开展上有很大的经营自主权，但是，在财务管理上集团希望通过集中信息处理方式进行有效监控，一方面可以规避和控制各种经营风险与管理漏洞，另一方面便于集中整体的资金优势开拓业务。集团企业进行集中财务管理的需求也越来越迫切。

集中财务管理解决的问题如下：

（1）建立能够实现多级法人、多级责任中心，多维核算记账体系，从而从多方面分析各种财务信息，例如，考核一个销售部门（利润中心），不仅要分析他们的三大财务报表，同时还要更精确的分析哪些客户、哪些产品、哪些区域带来的收入与利润是多少，其收入、利润构成结构是怎样的，在哪些客户身上的投入获得了多少回报，值不值？如果没有精细化、多角度的财务分析数据，经营决策常常会出现偏差给企业带来风险。

（2）与业务密切结合的预算，才能充分适应业务变化对资金需求的变化。企业的计划是随市场变化不断进行滚动的，第一节中提到了生产计划要紧随市场的变化，而财务计划——预算，要支持这些业务的变化。这样才能持续地适应市场。

（3）对于拥有众多分子机构的企业集团，财务合并报表制作的周期要长，以联想集团为例，在实施 ERP 系统前，合并报表周期为一个半月，之后为三天以内。很多企业常常拿着滞后的报表信息做决策，在应用 ERP 后将大有改观。

（4）许多大型企业集团的总部管理机关缺乏基层部门的业务信息，特别是财务信息的严密监控，ERP 系统通过集中记账，可以实现对财务凭证的实时查询，如果将财务与业务信息做到集成应用，就可从总公司的各种财务报表快速追溯到某个基层公司或孙子公司的业务订单的详细信息，依据这种平台的审计机制将是比较完美的。

现金流是企业的血脉，财务与业务的无缝集成使得业务发生对应的现金流入信息与现金流出信息能够快速在 ERP 系统中生成，从而提高企业对现金流的监控和预测水平。

15.3.2　财务管理软件的功能

财务管理软件的功能包括总账、应收账、应付账、工资管理、固定资产、财务报表、现金流量、资金管理、财务分析、网上银行、公司对账、预算管理、成本管理、项目成本。

1．总账

在财务体系结构中，总账（General Ledger）处于核心位置，每一个模块的财务信息最终都将汇总到总账中去，以便于编制综合的财务报表。总账用于各类企事业单位进行凭证管理、账簿处理、个人往来款管理、部门管理、项目核算和出纳管理等。可根据需要增加、删除或修改会计科目或选用行业标准科目。提供资金赤字控制、支票控制、预算控制、外币折算误差控制以及查看科目最新余额等功能，加强对发生业务的及时管理和控制，以及控制出纳科目、个人往来科目、客户往来科目、供应商往来科目。提供支

票登记簿功能，用来登记支票的领用情况；并可完成银行日记账、现金日记账，随时出最新资金日报表，余额调节表以及进行银行对账。自动完成月末分摊、计提、对应转账、销售成本、汇兑损益、期间损益结转等业务。进行试算平衡、对账、结账、生成月末工作报告。

2．应收账

应收账（Accounting Receivable）管理系统，管理企业在经营过程中所产生的各种应收款，通过发票、其他应收单、收款单等单据的录入或由销售管理自动生成单据，客户回款时，可将回款在应收单据之间分配，冲抵客户应收款，并自动计算现金折扣，生成收款凭证。对企业的往来账款进行综合管理，及时、准确地提供客户的往来账款余额资料，提供各种分析报表，如账龄分析表、周转分析、欠款分析、坏账分析、回款分析情况分析等，通过各种分析报表，帮助企业合理地进行资金的调配，提高资金的利用效率。在实际的经营活动中，企业与其他单位和个人发生的应收账款往往是比较频繁的，收款工作量大，拖欠款情况也时有发生，应收账款的管理是一项相当繁杂的工作。应收账系统可以帮助企业管理好应收款项，及时收回欠款，并有效地控制资金的使用，从而使企业获得最好的经济效益。

3．应付账

应付账（Accounting Payable）管理系统管理企业在经营过程中所产生的各种应付款数据信息，通过发票、其他应付单、付款单等单据的录入，对企业的往来账款进行综合管理，及时、准确地提供供应商的往来账款余额资料，提供各种分析报表，帮助您合理地进行资金的调配，有效地管理应付款，计划和控制资金的使用，提高资金的利用效率。

4．工资管理

工资管理（Wage Management）用于各类企业、行政事业单位进行工资核算、工资发放、工资费用分摊、工资统计分析和个人所得税核算等。可以与总账系统集成使用，将工资凭证传递到总账中；可以与成本管理系统集成使用，为成本管理系统提供人员的费用信息。

5．固定资产

固定资产（Asset Management）用于进行设备管理、折旧计提等，同时可为总账系统提供折旧凭证，为成本管理系统提供设备的折旧费用依据。帮助企业对固定资产进行卓有成效的管理，充分发挥资产的效能，最大限度地减少资金占用，保证企业生产经营活动的顺利进行。固定资产台账包括设置资产号、子号（用于资产的维修及更换零部件）、固定资产组、固定资产折旧类型、固定资产方位、固定资产的服务公司、租赁公司、固定资产各种价值（原值、现值、折旧、残值等）、固定资产的制造期、使用期、建设期等信息。对有关固定资产变动的历史进行跟踪能够按多种方法准确计算固定资产的折旧。

6．财务报表

根据各种财务信息定义和定制企业常用的财务报表（Financial Statements）。可提供

各行业报表模板，具备报表文件管理、格式定义、数据处理、图表分析、打印等功能。

7．现金流量

编制现金流量（Cash Flow Statement）表，提供企业一定期间内现金流入和流出的信息，以便了解和评价企业获得现金的能力。主要包括：自动生成的现金流量表及附表；对企业的现金流量进行按日、按月、按季、按年的准确反映；对已生成的期间现金流量表进行汇总；按企业特色设置的应收应付科目和增值税率，自动进行价税分离；对各种凭证提供了多种自动拆分方法或用手工拆分方式将现金流入项目与现金流出项目从凭证中分离出来并进行汇总统计；对于有外币核算的企业，可解决汇率变动对现金的影响。

8．资金管理

资金管理（Fund Management）系统为用户提供了全面、灵活、实用、准确的资金预测功能，通过资金预测，用户能随时掌握企业未来的资金流向、流量和盈缺情况；同时资金管理系统还提供了资金风险预警功能，以帮助用户防范支付危机；除此之外，用户还可以进行筹投资规划和筹资、投资管理。

9．财务分析

财务分析（Financial Analysis）是财务管理的重要组成部分，是利用已有的账务数据做进一步的加工、整理、分析和研究，从中取得有用的信息，对企业过去的财务状况、经营成果及未来前景的一种评价。系统运用对比分析、结构分析、绝对数分析、环比分析、趋势分析等多种分析方法，并进行预算自动编制，将预算与实际比较分析，分析内容包括：因素分析、基本财务指标分析、现金收支分析、现金收支增减分析、现金收支结构分析等方面。通过多种图形以直观、明了、易于理解的方式展现出来，从而为决策提供正确的、科学的依据。

10．网上银行

随着互联网技术的发展以及电子商务的兴起，国内各银行大都推出了网上银行业务。网上银行业务一般包括针对个人的网上个人银行业务和针对企业的网上企业银行业务两种。ERP 系统主要考虑的是网上企业银行业务。网上银行主要业务过程是企业将收款人信息提交给银行，银行按企业要求将款项由企业账户划转到收款人账户。本系统考虑了网上支付（即上传收款方信息）、交易信息查询下载（银行对账单下载、银行账户余额、银行账户交易明细查询）等业务，集团业务可以融合于其中。

11．公司对账

由于公司与公司之间在应收、应付账款的记账、结算上存在着时间差，因此为防止和及时发现并纠正可能出现的差错，应定期在有往来款项发生的公司之间核对账面余额，按月编制往来余额调节表，这就是公司对账（Company Audit）。

核对方法为：根据往来对账单与公司往来账逐笔核对发生额及余额，如发生与往来对账单不一致的记录，先检查双方是否记账错误，若是记账错误，则进行更正或通知对方单位更正，若不是记账错误，则为未达账项。“往来对账”包括集团内部单位间的往来

账和外部单位之间（客户、供应商）的对账。

12．预算管理

在缺乏信息化的手段下，预算制订不能够与其他业务流程相互集成，没有一个集成的信息系统使数据来源单一。由于缺乏一套全面集成的信息系统，编制预算的信息来源不够全面。预算的执行和跟踪的力度人为控制困难。

ERP系统可帮助财务部门根据企业的经营战略和相关历史数据，利用各种预算模型，制定实际可行的财务预算，确保预算的科学性、合理性与严肃性，达到预算管理（Budget Management）的目的。同时针对各子公司、各业务部门（责任中心）的具体情况，进行预算的分解，自上而下地制订部门预算，各业务部门的预算调整也会直接自下而上地汇总，影响总体预算情况。

ERP中还可维护多套计划预算数据，用于不同角度的分析评估。由于费用归集的实时集成，财务部门和管理层将能够随时监控预算计划执行情况，实现过程控制，确保企业财务计划的顺利执行，并根据用户自定义的控制标准对财务操作进行自动监控，对于超出预算计划的业务支出自动冻结并按系统设置向相关用户和管理人员发布警告信息。

13．成本管理

企业生存和发展的关键，在于不断提高经济效益。提高经济效益的手段，一是增收，二是节支。增收靠创新，节支靠成本控制。而成本控制的基础是成本核算工作。目前在企业的财务工作中，成本核算往往是工作量最大、占用人员最多的工作，企业迫切需要应用ERP系统来更加准确及时地完成成本核算工作。成本管理包括成本预测、成本计划、成本控制、成本核算、成本分析和成本考核等几个组成部分，其中，成本核算是成本管理的基础内容，是开展其他成本管理项目的前提。通过一个有效的手段来进行分析和控制合理的生产成本，可以降低成本，具体表现如下。

（1）控制消耗定额及支出。

（2）建立标准成本体制，责任清晰。

（3）成本计算方法先进，成本信息真实，商品定价有凭有据。

（4）基础管理扎实，实时得到准确及时的生产数据和整个供应链上协同。

14．项目成本

对于项目型企业项目成本的核算方式比较特殊，使用ERP系统可以追踪项目阶段、任务及成本支出的详细资料。这些详细资料包括所有项目的项目类型、管理部门、关键项目成员、分类及工作细目分类结构。ERP还可以管理企业的项目预算，并定义工作阶段以及细目分类结构，按模板建立项目以加速数据输入；定义项目类型、项目分类、费用控制、项目实施计划；分配项目中任务或作业的负责人员，及其他关键成员；对一段时期的或项目阶段成本及工时进行分析。

15.4　客户关系管理

CRM 是为企业提供全方位的管理视角，赋予企业更完善的客户交流能力，最大化客户的收益率。CRM 的焦点是自动化并改善与销售、市场营销、客户服务和支持等领域的客户关系有关的商业流程。CRM 既是一套原则制度，也是一套软件和技术。它的目标是缩减销售周期和销售成本、增加收入、寻找扩展业务所需的新的市场和渠道以及提高客户的价值、满意度、赢利性和忠实度。

CRM 应用软件将最佳的实践具体化并使用了先进的技术来协助企业实现这些目标。CRM 在整个客户生命期中都以客户为中心，这意味着 CRM 应用软件将客户当作企业运作的核心。CRM 应用软件简化协调了各类业务功能（如销售、市场营销、服务和支持等）的过程并将其注意力集中于满足客户的需要上。CRM 应用还将多种与客户交流的渠道，如面对面、电话接洽以及 Web 访问协调为一体，这样，企业就可以按客户的喜好使用适当的渠道与之进行交流。CRM 强调“以客户为中心”，除了把握客户真实的需求和快速响应、满足客户个性化的需求外，更为重要的是提高客户满意度和客户忠诚度。客户忠诚的前提是客户满意，而客户满意的关键条件是客户需求得到满足。

CRM 产品通常汇集了当今最新的信息技术，如数据仓库技术、网络技术、商业智能等，与市场营销等管理学科的管理理念。

15.4.1　客户关系模型

CRM 系统的字面意义“客户关系”则来自其管理系统的基础结构：客户关系模型，按照字面解释就是设计用来支持复杂的贸易社区的体系结构。客户关系模型的目标是把所有的“关系”包含在“贸易社区”中。例如，一个器具制造厂商的贸易社区，会包含供应商、分销商、零售商、服务提供商、个体客户和商务客户。这个器具制造厂商不仅希望跟踪自己和贸易社区中其他实体的关联，而且希望了解贸易社区中各个成员之间的关联。这个器具制造厂商和贸易社区中的所有成员不一定都有直接的关联。但是，对于这个器具制造厂商来说，了解贸易社区中各个成员之间的关联是非常重要的。

在客户关系模型中，“关系”的类型主要有如下三种：

（1）组织与组织的关系。例如，母子公司间的关系、贸易伙伴间的关系、客户与供应商的关系等。

（2）组织与个人的关系。例如，公司与职员的关系、自然人与企业的投资关系、公司与个人客户的关系等。

（3）个人与个人的关系。例如，父子关系、上下级关系、同事关系、同学关系。

而每一个组织和个人构成了关系模型中的一个个节点，每一个节点又有大量的属性管理信息。对于组织，有财务信息、联络信息等；对于个人，有年龄、工作、联络、嗜

好等信息的管理。

1．成员概念

成员的概念使客户关系模型能够平等地对待所有商业实体，无论是怎样类型的客户。平等对待所有商业实体，客户模型就能够很容易地处理组织对组织（Business To Business，B2B），组织对个人（Business To Consumer，B2C）或混合的商业模式。虽然要面对个人和组织的不同属性：对于个人，是出生日期、职务及性别，而对于组织，则是企业登记号、税务登记证号及财年结算日。无论个人还是组织都是成员，都存储在数据库的同一个表中。这样，某一类型的组织之间的商业关系可以很容易地联系到个人。将多个成员组合可形成一个单一的实体——组。一个组是一个单独的实体，同时组中的成员也是独立实体。关系类型成员允许两个成员的关系被看成是一个在其自己权限内的成员。例如，王勇在希赛 IT 教育研发中心工作，这其中有三个成员，分别是王勇（个人类型成员）、希赛 IT 教育研发中心（组织类型成员）、王勇在希赛 IT 教育研发中心（关系类型成员）。因为王勇在希赛 IT 教育研发中心是一个有其自己权限的成员，其地址、电话、客户号等信息都可以直接与该实体关联。

2．成员与地址的多对多关系

成员与地址的多对多关系的管理降低了重复录入，加强了大量客户关系信息的有效共享。由于一个地址关联于几个成员，地址本身就不必重复多遍。而且，如果地址改变，只需更新一次。

3．高级关系模型

在客户关系模型中，成员的关系存储在不同的实体内，这样，任何成员都可以是任意数量成员关系中的成员。另外，由于每个成员的关系是用关系类型来定义的（如某人的父母、某公司的员工、组织的分支、某原厂商的代理等），两个成员可以具有多重关系（如张明是李红的未婚夫，同时也是李红的商业伙伴）。客户模型支持当前及历史关系。因为成员关系存储在不同的表中，它们可以有自己的属性。两个主要的保存关系信息的属性是“起始日期”及“结束日期”。当一个成员关系结束，用户将提供结束日期而不是删除它。因此，宝贵的关系历史被无限期地保留下来。

4．成员与关系分离

客户关系模型分离非常重要，因为该分离允许成员可以有多个商业关系。其他成员也可以与该“关系”关联，例如，某个“关系”的授权购买者及担保人。

5．灵活的成员分类

客户关系模型允许任意自定义成员分类，可以按行业、数量及购买习惯将客户分类。该用户不仅决定目录，还可以决定目录下的类别或数值。

对于大型服务型企业（如保险公司）有着重大的意义，保险公司的业务按客户类型划分为团险（针对组织）和个人险。保险营销需要不断挖掘潜在价值客户，而 CRM“客户关系模型”管理可以将分散在企业各个部门、各个销售人员的信息进行汇总、分析和

深度挖掘，并归档。对于一家保险公司，一个销售人员的流失给企业带来多少客户关系的丧失？CRM 可以帮助企业降低因人员流失导致的客户关系的损失，不仅如此，如果保险公司根据大量客户数据的分析，如个人嗜好、收入水平、年龄等信息的分析，可以更有效的推出新的保险产品，以赢得竞争优势。

15.4.2　CRM 的功能

CRM 系统主要由市场管理、销售管理、客户服务管理构成。

1．营销智能

营销智能（Marketing Intelligence）通过分析、汇总和统计来自于 CRM 系统和业务系统的市场、客户、交易与服务信息，为企业高层管理者提供辅助决策信息。营销智能也称为销售智能（Sales Intelligence）、客户智能（Customer Intelligence）。

2．市场营销

市场营销（Marketing Online）帮助企业全面控制和管理各种市场活动、促销活动。市场活动从计划、预算到执行，以及结果的反馈与记录，可以进行全程的控制。使企业达到以最小的投入获得最大的市场推广效果，帮助企业准确进行市场定位、产品定位。

3．电话销售

电话销售（Tele-Sales）将客户的各方面信息纳入系统化的管理，客户信息包括通信、社会关系、兴趣爱好、对产品的认知情况，收入及服务状况等等。使用“电话销售”，企业将全面覆盖目标市场的详细状况，与“市场营销”结合起来能更有效地影响目标客户，对细分市场客户做到准确分析与把握。

4．在线销售

在线销售（Sales Online）帮助企业控制项目销售的全部过程。在线销售包括：将销售线索转化为销售机会，销售人员进行销售并最终进行报价，汇总各级销售组织的信息，按期间进行销售预测等方面。通过它，经理可以了解每个销售队伍、销售人员的业绩和工作成果，每个销售机会的状态（刚接触、已确认、签约、已回款等）、客户信息（通信地址、决策人等）、成功的概率、预计收入、销售产品等等，销售人员可以时刻更新这些信息。此外，经理可以按地区、客户、产品、销售等方面进行统计，分析最有潜力的客户、产品等。它使得公司能够更有效的控制销售、把握市场。

5．销售佣金管理

销售佣金管理（Incentive Compensation）帮助集成企业建立合理的奖励体系，充分调动销售人员的积极性。

6．网上商店

网上商店（e-Store）提供产品目录管理、促销管理等功能，根据业务变化，自动生成相应的 Web 网页，帮助企业以低成本建立网上商店，拓展业务。

7. 在线服务

在线服务（Service Online）帮助企业快速响应客户服务请求，协助企业协调服务人员及时提供现场服务，以提高客户的满意度。比如，客户打电话询问有关产品升级的情况，呼叫中心系统根据客户拨入的电话号码自动检索客户档案，系统将客户以往的购买情况、产品维修情况、联系人、本次请求的内容（如果以 Internet website 访问的方式）等信息全部显示出来。客服人员可以根据客户要求按一定条件（如地区、工程师类型等）查询符合条件的售后支持人员并进行委派，以快速提供支持服务。

8. 网上服务

网上服务（e-Support）帮助企业建立类似于 Helpdesk 的网站，通过计算机网络系统，自动处理客户对产品与技术方面提出的问题，使客户通过网络自助服务，为企业降低服务成本与服务资源的占用。

9. 交互中心

交互中心（Call Center）自动应答客户来电，可以提示接线人员有关的客户档案，历史交易、维护记录，包括 Inbound（呼入）、Outbound（呼出）、Scripting（脚本编制）、E-mail Center（电子邮件中心）和 Interaction Center Intelligence（交互中心智能）几个部分，它为企业提供了企业与客户交互沟通的基础平台。

15.5 产品生命周期管理

产品生命周期管理（Product Life-cycle Management，PLM）是一个集成的、信息驱动的方法，它由人、过程/实践和技术组成，应用于从设计、制造、配置、维护、服务到最终处理的产品生命周期的所有方面。PLM 系列软件能够存取、更新、处理和推理由局部的和分布环境中产生出来的产品信息。

著名咨询公司 AMR 认为，产品生命周期管理是一种技术辅助策略，把跨越业务流程和不同用户群体的那些单点应用集成起来。AMR 把 PLM 的内容大致分为 4 个应用部分。

（1）产品数据管理（Product Data Management，PDM）：起着中心数据仓库的作用，它保存了产品定义的所有信息。从这些中心仓库，企业管理各类的与研发和生产相关联的 BOM。

（2）协同产品设计：让工程师和设计者使用计算机辅助设计等软件以及所有与这些系统配合使用的补充性软件，以协同的方式在一起研发产品。

（3）产品组合管理：是一套工具集，它为管理产品组合提供决策支持，包括新产品和现有产品。产品组合管理工具集有三个部分：用于日常工作任务协调的项目管理；用于一次处理多个项目的纲要管理；用于理解产品如何共存于市场的组合管理。

（4）客户需求管理：是一种获取销售数据和市场反馈意见，并且把它们集成到产品

设计和研发过程之中的软件。正如在名称上所体现的，它是一个分析工具，可以帮助制造商开发基于客户需求、适销对路的产品。

当然，只有这 4 个部分还不足以组成 PLM，这只是 PLM 4 个主要的应用部分。

产品生命周期是在 2000 年初被提出的，计算机辅助设计（Computer Aided Design，CAD）、工程数据管理（Engineering Data Management，EDM）、PDM 和计算机集成制造（Computer Integrated Manufacturing，CIM）等是 PDM 形成的基础。

1．计算机辅助设计

CAD 的概念和内涵还在不断地发展之中。CAD 是一种技术，其中人与计算机结合为一个问题求解组，紧密配合，发挥各自所长，从而使其工作优于每一方，并为应用多学科方法的综合性协作提供了可能，CAD 是工程技术人员以计算机为工具，对产品和工程进行设计、绘图、分析和编写技术文档等设计活动的总称。

根据模型的不同，CAD 系统一般分为二维 CAD 和三维 CAD 系统。二维 CAD 系统一般将产品和工程设计图纸看成是“点、线、圆、弧、文本”等几何元素的集合，系统内表达的任何设计都变成了几何图形，所依赖的数学模型是几何模型，系统记录了这些图素的几何特征。二维 CAD 系统一般由图形的输入与编辑、硬件接口、数据接口和二次开发工具等几部分组成。三维 CAD 系统的核心是产品的三维模型。三维模型是在计算机中将产品的实际形状表示成为三维的模型，模型中包括了产品几何结构的有关点、线、面、体的各种信息。由于三维 CAD 系统的模型包含了更多地实际结构特征，使用户在采用三维 CAD 造型工具进行产品结构设计时，越能反映实际产品的构造或加工制造过程。

2．工程数据管理

EDM 用于产品的数据收集、跟踪和报表的专用程序，工程技术人员也常用 Microsoft Excel 来完成这些工作。

产品的数据是指描述产品的任何信息，包括公差、抗拉强度、重量限制、粘合剂和传导率要求等。也包括用于描述产品本身的信息，如制造产品的过程、包装和为产品喷漆的方法、测试产品的方法、测试所用的仪器以及测试的结果。

3．产品数据管理

PDM 是企业为了对不断增加的 CAD 文件进行组织和分类而产生的，一个主要目的是将企业存储在纸质的和胶片中的文档转换成电子文档。PDM 是一门用来管理所有与产品相关信息和所有与产品相关过程的技术。与产品相关的信息包括产品结构和配置、零件定义及设计数据、CAD 绘图文件、工程分析及验证数据、制造计划及规范、数控编程文件、图像文件（如照片、造型图、扫描图等）、产品说明书、软件产品（如程序、库、函数等“零部件”）、各种电子报表、成本核算、产品注释等、项目规划书、多媒体音像产品、硬拷贝文件、其他电子数据等。与产品相关的过程包括过程定义和管理。

PDM 的使用者并不局限在产品研发和设计部门，在企业内，只要是与产品数据打交

道的人，都可以使用 PDM。在企业的信息集成过程中 PDM 系统可以被看作是起到一个集成框架的作用，各种应用程序将通过各种各样的方式（如应用接口、开发（封装）等），直接作为一个个对象而被集成进来，使得分布在企业各个地方、在各个应用中使用（运行）的所有产品数据得以高度集成、协调、共享，所有产品研发过程得以高度优化或重组。

4．计算机辅助制造

计算机辅助制造（Computer Aided Manufacturing，CAM）是指利用计算机来进行生产设备管理控制和操作的过程。它输入信息是零件的工艺路线和工序内容，输出信息是刀具加工时的运动轨迹（刀位文件）和数控程序。

5．计算机集成制造系统

计算机集成制造系统（Computer Integrated Manufacturing System，CIMS）是随着计算机辅助设计与制造的发展而产生的。它是在信息技术自动化技术与制造的基础上，通过计算机技术把分散在产品设计和制造过程中各种孤立的自动化子系统有机地集成起来，形成适用于多品种、小批量生产，实现整体效益的集成化和智能化制造系统。集成化反映了自动化的广度，它把系统的范围扩展到了市场预测、产品设计、加工制造、检验、销售及售后服务等的全过程。智能化则体现了自动化的深度，它不仅涉及物资流控制的传统体力劳动自动化，还包括信息流控制的脑力劳动的自动化。

在我国，CIM/CIMS 又被赋予了新的意思，现在意指现代集成制造（Contemporary Integrated Manufacturing）与现代集成制造系统（Contemporary Integrated Manufacturing System）。它已在广度与深度上拓展了原 CIM/CIMS 的内涵。其中，“现代”的含义是计算机化、信息化、智能化。“集成”有更广泛的内容，它包括信息集成、过程集成及企业间集成等三个阶段的集成优化；企业活动中三要素及三流的集成优化；CIMS 有关技术的集成优化及各类人员的集成优化等。

15.6 企业信息化的其他内容

在企业信息化过程中，根据企业的不同，可能会有一些不同的系统，包括办公自动化系统、制造执行系统、人力资源管理系统、企业资产管理、商业智能、企业门户、电子商务、协同商务、企业应用集成、业务流程管理、知识管理等。

1．办公自动化系统

名字还是那个名字，但其包含的内容在发生着不断的变化。20 世纪 80 年代谈到办公自动化（Office Automation，OA），不过是一个文字处理工具，现在讲到 OA，通常已经是融入了知识管理、工作流技术，充分应用 Internet 环境，集成移动通信技术，使组织内部员工充分共享信息，高效协同工作，实现快速、全方位的信息采集，信息处理，为管理和决策提供科学的依据。

2．制造执行系统

制造执行系统（Manufacturing Execution System，MES）是美国管理界在 20 世纪 90 年代提出的新概念，近 20 年来在国际上迅速发展。它是面向车间层的生产管理技术与实时信息系统。MES 可以为用户提供一个快速反应、有弹性、精细化的制造业环境，帮助企业减低成本、按期交货、提高产品的质量和提高服务质量。适用于不同行业（如家电、汽车、半导体、通信、IT、医药），能够对单一的大批量生产和既有多品种小批量生产又有大批量生产的混合型制造企业提供良好的企业信息管理。目前国外知名企业应用 MES 系统已经成为普遍现象，国内许多企业也逐渐开始采用这项技术来增强自身的核心竞争力。MES 的定位，是处于计划层和现场自动化系统之间的执行层，主要负责车间生产管理和调度执行。一个设计良好的 MES 系统可以在统一平台上集成诸如生产调度、产品跟踪、质量控制、设备故障分析、网络报表等管理功能，使用统一的数据库和通过网络联接可以同时为生产部门、质检部门、工艺部门、物流部门等提供车间管理信息服务。系统通过强调制造过程的整体优化来帮助企业实施完整的闭环生产，协助企业建立一体化和实时化的现场楼层控制系统（Shop Floor Control System，ERP/MES/SFC）信息体系。

3．人力资源管理系统

人是组织最重要的资产和资源，组织间的竞争也是人力资源的竞争，从这个意义上讲，组织必须做好人力资源的开发和利用。人力资源管理系统（Human Resource Management System，HRMS）从组织的管理架构设计开始，定义每一个岗位，定义每一个岗位的任职资格，对岗位的任职资格用量化的能力来定义。

组织中的每一个人都有丰富的属性定义，对其能力有量化的核定。可以测量每一个员工适合的岗位，也可以测量他在现任岗位欠缺的能力。培训是提高员工能力的手段，可以提供在线的培训，也可以组织管理面对面的内部培训或外部培训，通过培训后的考核来确定能力的提高。通过对员工职业规划的管理，可以给员工努力的方向，也可以了解员工的志向。对员工的考评是多维度的，不仅来自于他的直接上司，还来自于他的下属、他的同级同事、他的合作伙伴、他服务的内部或外部客户，也有来自于他的业绩方面的数量指标，每个方面的考核在总体考核中的权重是可以定义的。HRMS 系统应该能够提供员工对个人基本资料的维护功能，提供招聘管理、时间管理、薪酬管理、福利管理、培训管理等功能。能提供对人力资源指标的智能分析。

4．企业资产管理

企业资产管理（Enterprise Asset Management，EAM）是面向资产密集型企业的企业信息化解决方案的总称。它以提高资产可利用率、降低企业运行维护成本为目标，以优化企业维修资源为核心，通过信息化手段，合理安排维修计划及相关资源与活动。通过提高设备可利用率得以增加收益，通过优化安排维修资源得以降低成本，从而提高企业的经济效益和企业的市场竞争力。在商业竞争日益激烈的今天，对于拥有高价值资产的企业来说，设备维护已不再局限于成本范畴，更成为获取利润的战略工具，EAM 系列产

品使这一目标得以实现。

EAM 是以企业资产及其维修管理为核心的应用软件套件，它主要包括基础管理、工单管理、预防性维护管理、资产管理、作业计划管理、安全管理、库存管理、采购管理、报表管理、检修管理、数据采集管理等基本功能模块，以及工作流管理、决策分析等可选模块。

EAM 以资产模型、设备台账为基础，强化成本核算的管理思想，以工单的创建、审批、执行、关闭为主线，合理、优化地安排相关的人、财、物资源，将传统的被动检修转变为积极主动的预防性维修，与实时的数据采集系统集成，可以实现预防性维护。通过跟踪记录企业全过程的维护历史活动，将维修人员的个人知识转化为企业范围的智力资本。集成的工业流程与业务流程配置功能，使得用户可以方便地进行系统的授权管理和应用的客户化改造工作。

5．商业智能

商业智能是帮助企业更好地利用数据提高决策质量的技术，包含了从数据仓库到分析型系统等。这些分析有财务管理、点击流（clickstream）分析、供应链管理、关键绩效指标（Key Performance Indicators，KPI）、客户分析等。商业智能关注的是，从各种渠道（如软件、系统、人等等）发掘可执行的战略信息。商业智能用的工具有抽取、转换和加载软件（搜集数据，建立标准的数据结构，然后把这些数据存在另外的数据库中）、数据挖掘和在线分析等。商业智能是企业信息化比较高层次的应用，为企业的决策和战略管理提供支持。有关商业智能的详细知识，请参考本书 6.1 节。

6．企业门户

Portal（门户）是企业信息化中一个集中访问的解决方案。给用户一个统一的访问界面，因为每一个应用系统都有它的局限性，高度信息化的企业，每一个用户在工作中都会用到多个应用系统，反复地切换系统，记住每个系统的登录账号和密码，那是繁琐而辛苦的，使用 Portal 后，把应用系统包起来，把真正的应用系统隐藏在后台，每个用户使用一个账号就可以使用自己权限内的所有系统的相关功能，把这些功能菜单集成到一个网页上，这个网页是个性化的，每个员工可以定义自己的网页格式，在自己的网页界面上会提示他现在需要做的工作，可以展现公司的内部通告和新闻等。

归根结底，Portal 就是一个内容聚集和展现的窗口。Portal 也分很多类别，分为基于内容的 Portal、个性化 Portal、企业级 Portal。

7．电子商务

顾名思义，电子商务（Electronic Commerce）是指采用数字化电子方式进行商务数据交换和开展商务业务的活动。从广义讲，电子商务还包括企业内部商务活动，如生产、管理、财务以及企业间的商务活动。它不仅是硬件和软件的结合，更是把买家和卖家、厂家和合作伙伴在 Internet 上利用 Internet 技术与现有的系统结合起来进行业务。从最初的电话、电报到电子邮件以及 20 多年前开始的 EDI，都可以说是电子商务的某种形式。

发展到今天，人们已提出了包括通过网络来实现原材料的查询、采购，产品的展示、订购到出品、储运以及电子支付等一系列贸易活动在内的完整电子商务的概念。基于电子商务推出的金融电子化方案、信息安全方案、Internet 方案等形成一个又一个产业，给信息技术带来许多新的机会。

对于电子商务概念的科学理解应包括以下几个基本方面：电子商务是整个贸易活动的自动化和电子化；电子商务是利用各种电子工具和电子技术从事各种商务活动的过程。其中，电子工具是指计算机硬件和网络基础设施，电子技术是指处理、传递、交换和获得数据的多技术集合。电子商务渗透到贸易活动的各个阶段，因而内容广泛，包括信息交换、售前售后服务、销售、电子支付、运输、组建虚拟企业、共享资源等等。电子商务的参与者包括消费者、销售商、供应商、企业雇员、银行或金融机构，以及政府等各种机构或个人。

电子商务的目的就是要实现企业乃至全社会的高效率、低成本的贸易活动。电子商务从交易对象上区别，有 B2B 模式、B2C 模式、消费者对消费者（Consumer to Consumer，C2C）模式。

8．协同商务

协同商务（Collaboration Business）在 2000 年初由 Gartner Group 提出。协同商务是指在全球经济一体化的背景下，利用以 Internet 等为特征的新兴技术为实现手段，在企业的整个供应链内及跨供应链进行各种业务的合作，最终通过改变业务经营的模式与方式达到资源最充分利用的目的。按照领域的不同大致分为设计、商务与制造三个环节。新的业务模式主要是虚拟业务（Virtual Business），虚拟业务的组成部分正好对应于上面提到的三个环节，分别是虚拟设计、虚拟商务与虚拟制造。

9．企业应用集成

EAI 包括业务过程集成、应用集成、数据集成、界面集成等多方面，能够提升企业整体运营效率，为企业带来新的发展动力。伴随着 EAI 技术的不断发展，它所被赋予的内涵变得越来越丰富。起初，EAI 只是指企业内部不同应用系统之间的互连，如今谈到 EAI 的概念，具有更为广义的内涵，它已经被扩展到业务整合的范畴。

业务整合不仅要提供底层应用支撑系统之间的互连，同时要实现存在于企业内部应用与应用之间，本企业和其他合作伙伴之间的端到端的业务流程的管理，它包括应用整合、B2B 整合、自动化业务流程管理、人工流程管理，企业门户以及对所有应用系统和流程的管理和监控等方方面面。具体到技术层面上的划分，一套完整的 EAI 技术层次体系应该包括应用接口层、应用整合层、流程整合层和用户交互层 4 个大的层面。有关 EAI 的详细知识，请参考本书第 9 章。

10．业务流程管理

业务流程管理（Business Process Management，BPM）是一套达成企业各种业务环节整合的全面管理模式。BPM 涵盖了人员、设备、桌面应用系统、企业级 Backoffice 应

用等内容的优化组合，从而实现跨应用、跨部门、跨合作伙伴与客户的企业运作。BPM 通常以 Internet 方式实现信息传递、数据同步、业务监控和企业业务流程的持续升级优化。显而易见，BPM 不但涵盖了传统“工作流”的流程传递、流程监控的范畴，而且突破了传统“工作流”技术的瓶颈，是工作流技术和企业管理理念的一次划时代飞跃。

11．知识管理

简单地说，知识管理（Knowledge Management，KM）就是企业对其所拥有的知识资源进行管理的过程，而如何识别、获取、开发、分解、储存、传递知识、从而使每个员工在最大限度地贡献出其积累的知识的同时，也能享用他人的知识实现知识共享则是知识管理的目标。

15.7 信息化项目实施的风险和控制

在知识经济的今天，效率和成本已经成为公司生存和发展的根本要素，企业管理的信息化已经成为现代企业的必然选择。信息化无法躲避，只能去探讨如何减低风险，提高项目的应用效果。

企业信息系统的应用大致分为 IT 咨询、项目准备、系统建设、系统交付与持续改善这几个方面。在此期间，服务商和用户双方有效的沟通与密切的配合至关重要，用户方高层领导的高度参与尤其重要，双方的敬业精神与合作诚意是项目成功的前提。

15.7.1 来自人的风险和规避

一个被叫得很熟的口号是“一把手工程”，足以见企业一把手在项目中的重要意义。但一把手在项目中扮演怎样的角色，参与多深，对项目的成败影响很大。实践证明，一把手不可能有充足的精力参与到项目中去，他只能扮演项目的战略决策者、仲裁人和激励者，项目的真正领导者和推动者应该是紧跟在他后面的经他充分授权的一个人。如果过分地强调一把手在项目中的重要性，一把手又真正地要身先士卒，如果项目中的很多事都要一把手来参与决定的话，一把手很可能成为项目进程中的“瓶颈”，很多任务会因为他的时间的冲突而进展不下去。一把手是一个企业的灵魂，但凡涉及到企业伤筋动骨的项目都要他来拍板，都是所谓的“一把手工程”。所以对“一把手工程”的理解不能片面，过于片面会走向另外一个极端。

项目经理是项目的具体领导者和执行者，通常用户方和咨询服务商各安排一名项目经理。项目经理的称职与否对项目影响很大。项目经理在项目中主要责任是项目的控制、协调和指挥。沟通能力、控制能力对项目经理来讲是至关重要的两项能力。责任心、敬业精神是项目经理必须具备的品格，兼听并蓄是项目经理必须遵从的工作方式。项目经理在项目中处于承上启下的位置。承上，接企业的最高管理者，要充分理解企业最高管理者对企业发展的战略；启下，要把企业领导者和项目委员会的意图完整地传导给项目

的执行者。项目经理要有很强的执行力，切忌老好人来做项目经理，也切忌项目经理成为“海绵垫”，如果项目中的很多矛盾到项目经理这里都缓解了，但不能从根本上解决，就为项目的崩溃留下了蚁穴。

每在项目启动前，用户方组建项目小组都是一件很难的事，难的是如何把人抽出来。的确，今天的企业都是一个萝卜一个坑，没有闲人。但所有经历过的项目都告诉笔者，企业信息化的项目是一个永续的工程，绝非一劳永逸的阶段性工作，所以，一定要在项目进行过程中培养出企业自己的，能保证后期应用推广和深入应用的人才。作为用户方项目组的成员，通常被称为“关键用户”，来自于企业的业务骨干和技术骨干，需要能够“讲出来，做下去”。参与项目的业务骨干需要能把目前的业务流程和业务需求讲清楚，能够理解供应链的管理思想，能有企业全局的观念，有一定的计算机应用经验，积极参与项目小组的方案讨论，最终有能力向业务部门解释项目小组设计的方案，有能力推动项目的有效执行。参与项目的技术骨干要有能力学会系统应用的日常维护和后续的外围开发。用户在组建项目组的时候，既要考虑到参与项目的员工的稳定性，也要考虑到项目组成员的学习能力和创新能力。

更准确地讲，信息化项目不仅是一个“一把手工程”，更是一个全员项目。几乎关系到企业中的每一个员工。要涉及到观念的转变、工作方式的转变、流程的转变。在项目的实施与推广过程中，参与项目的绝不仅仅是项目小组的几个人，而会是更多的人。他们需要接受理念的培训、流程的培训、系统应用的培训，他们还要参与整理数据。在项目中要调动每一个参与者的积极性，不妨参照马斯洛的需求模型（生理→安全→社会性→被尊重→自我实现），循循利诱，让每个人都发自内心积极地参与工作，让他们能感到项目推广过程中给他们自身带来的实惠，例如，工作能力的方面、事业发展的方面、企业发展后他们的个人收入方面等。简单的说教和施压往往适得其反，表面文章做好了，但实际应用的时候就掉链子，在数据整理和上线前培训往往会暴露问题。

外部顾问的问题也是一个关键风险所在，也是企业最关心的问题和难以回避的问题。在选型和选顾问公司的时候，用户方永远处于信息不对称的弱势那一方，即使选型过程拖得很长也难得要领，虽然在这个阶段接触很多供应商，但公说公有理，婆说婆有理，得到的信息越多，越是难以决策。

笔者认为，产品方案的决策倒是简单一些，从产品提供商的实力、发展和行业应用与产品说明可以有一个简单客观的选择。产品是相对静态的，在实施过程中变数不大。但实施的质量更多来自于直接提供服务的顾问。对顾问公司的考察包括公司本身和项目顾问两个方面。顾问公司是一个典型的知识密集型组织，它的管理不同于传统的制造型组织。决定顾问公司的稳定性主要取决于两个方面，一个在现金流，另一个在员工对组织的认同度，即企业文化。从顾问公司的人员规模和正在服务的项目规模可以大概考察到顾问公司的现金流是否正常，从与顾问公司销售、顾问和高层的反复接触，可以感觉到组织是否有一致的愿景。从顾问公司项目的管理方式，知识管理体系可以看到这家公

司是在谋取长远发展还是追求短期利益。在参观样板案例的时候邀见相关顾问可以了解顾问的稳定性。顾问个人的考察，应着重于行业经验和项目管理经验，要沟通顺畅、思路清晰、有组织能力、有执行力、有说服力。

15.7.2 来自流程的风险和规避

必须要认识到，ERP/CRM/SCM 等都是管理模式的概念，而不是一个软件的名字。企业在引进这些软件系统时，必须对管理模式的转变做好准备。新瓶装旧酒，换汤不换药的做法都会对软件系统的应用效果大打折扣。

早几年，提到 ERP，必谈业务流程再造/业务流程重组（Business Process Reengineering，BPR）。但由于再造的难度大和失败的案例多，以后就有人不提 BPR 了，改提业务流程改进/业务流程优化（Business Process Improvement，BPI）。但在实施过程中往往走到另一个极端，无论用户还是顾问公司都尽量回避“BP”，这就又错了。

企业在实施信息化以前，信息不共享，无论是有部门级的信息系统也好，还是完全的手工单证管理也好，部门间都存在信息壁垒，形成一个个的信息孤岛。组织架构以职能建立，多是树形的科层制组织结构。部门间不仅存在信息的壁垒，权利也存在壁垒，部门利益主义横行，跨部门的工作协调的工作量很大，甚至协调的难度也很大。

以 ERP 为例，ERP 的设计基于信息共享平台，强调物料流、资金流和信息流的统一。基于全程供应链的管理思想，供应链管理思想要求组织结构是跨职能的水平型结构。供应链管理须要协调大范围内的活动与流程。这些活动与流程往往跨越职能部门，包括采购和发料、进出口运输、接货、物料处理、存储和配送、库存控制与管理、供需计划、订单处理、生产计划、运输、加工，以及客户服务等。这些活动被看作是一个前后关联的系列，它们需要整合、协调与同步，这就是 SCM 的本质。所以，在 ERP 项目的推进过程中，要让用户彻底理解 SCM 的实质，接受 SCM 的思想，要让用户意识到他们每天从事的工作是一系列关联的活动，而不是分散的、孤立的活动。如果认识不到这一点，在涉及到跨部门或跨职能边界寻求效率改进的机会时，就总会遭遇到来自这些部门的直接阻挠。但十分遗憾的是，在很多 ERP 项目的实施过程中把这个环节忽略了。

德勤顾问公司（Deloitte Consulting）的最新调查指出，北美 91%的制造商将 SCM 列为最关键或非常重要的企业成功因素。

当然，变革需要条件和机会。我们相信，在项目推进过程中，为了降低项目的风险，首要的是 SCM 思想的普及与渗透，在管理层与执行层完全理解的基础上推进变革。变革是一个持续的过程，不是一蹴而就，也不是一夜间的变化。变革的动力要有持续的支持。

希赛教育专家提示：信息系统跑起来只是第一步，用得好需要“形”和“神”的统一。

15.7.3 来自项目管理的风险和规避

作为一个项目，其目标和范围首先必须是明确的。但实际的信息化项目往往又是模糊的，用户和项目咨询顾问往往都存在侥幸心理，一方想稀里糊涂地多要点，另一方却想稀里糊涂地少做点。这都会给项目带来风险。项目启动之后，合作双方已经完全被绑在了一条船上，没有你我之分，成功与失败对双方都一样重要，这个时候，双方要实事求是地评估项目资源（即投入项目的资金、时间和人力），制定切实可行的项目目标，圈定切实可行的实施范围。很多项目在完成之后也无法明确项目的效果，缺乏项目实施前后量化的指标比较。因此，在项目启动之初，双方要尽可能收集业务过程中的可测量的指标，如库存周转率、库存资金占用、按期交货率、一次检验合格率、客户投诉率、财务报告时间等。

项目组织的完整性是对项目成功的保障。通常需要有合作双方高层参加的项目领导小组，负责保证和协调项目所需的资源，决策项目相关的重大变革。双方合作的项目小组是项目推进过程中的常设机构，必须确保项目小组中人员的稳定性和对工作的胜任。

跨业务部门的信息化项目不同于小型的业务管理系统，在实施过程中还会牵扯到组织和流程的变化，甚至权力和义务的重新分配，所以说，信息化项目的推导是一个复杂的系统工程。把一个对开的年画挂在墙上是一件容易的事情，相信十几岁的孩子都可以做到，但是要把一幅几十平方米的广告画平整地挂在广告牌上，就不是一件容易的事，既需要技术也需要方法。顾问公司在帮助用户实施信息化项目的时候，既在项目方案方面给客户带来帮助，在项目实施方法和项目管理方面带来的价值也不可小觑，事实上，后者对把控项目的质量和风险更加重要。

各家主流的软件厂商都在不厌其烦地推介自己的所谓实施方法论，但真正理解项目管理精髓的项目经理仍然还是今天社会的稀缺资源。项目实施方法不是一成不变的，笔者认为，项目实施方法一定会依据项目对象（用户企业的规模和用户方项目组成员的素质因素等）、项目范围、项目实施周期、顾问水平等作适当的调整，不可能形而上学，一成不变。重要的是抓住项目管理的核心要素，即范围、时间、质量、成本。

一个客观可行的项目计划非常重要。项目上线时间及系统切换时间的宣布一诺千金，绝不可以轻易变更，特别是延期变更。只要有一次有理由的延期，项目组成员和最终用户就会对项目计划不以为然，他们会相信做不好工作一定还可以有下一次的延期上线的决定。笔者不建议在安排项目计划的时候，留出所谓的 Buffer，项目计划中有了预留的宽容，会给计划的严格执行留有余地，从项目一开始就给项目组留下了讨价还价的空间。在项目上线时间临近，资源又不能充分保证的时候，宁肯砍项目上线的范围，也要保证项目上线的准时。

有效的项目管理还必须要做到有效控制用户对项目的关注焦点。项目的成功与否，着眼点应该在全局，而不在局部，绝不可以因个别用户或个别顾问的过于专注把项目带

入局部的精雕细刻。一个项目的推进，好比给一个木箱装箱盖，箱盖 4 个角的螺丝要轮番循序渐进地拧，绝不可以把其中一个拧紧了才去拧第二个，那样做，无法把箱盖平平整整地拧上去，拧到最后还不得不把已经拧紧的螺丝再退两圈下来，搞不好还会把箱盖拧裂。

15.7.4 来自数据的风险和规避

人们常说两句话，一句是“三分软件，七分实施，十二分的数据”，另一句是“垃圾（数据）进，垃圾（数据）出”，足以见数据对项目的重要性。

项目实施过程中，项目咨询顾问要在方案确定后，就应该尽早与客户讨论数据格式、编码规则等规范，帮助客户制订静态数据的收集模版，并定期跟进客户的收集过程，确保按计划完成。

在项目上线前的动员会上，要再次向全体用户重申数据的重要性，强调数据录入的及时性和准确性。上线初期，项目组对每日的数据的准确性都要做检查，要写出数据差异报告。随着数据准确率的提高，数据差异报告的发布周期可以改为一周。必要时，系统数据差异的总结可以成为公司周例会的一个内容，充分加强各部门主管对数据准确性的认识。不少的企业在系统上线初期，甚至采用奖惩手段来确保系统采集数据的准确性。

本章参考文献

[1] SAP 公司. SAP R/3 产品技术白皮书
[2] Oracle 公司. Oracle e-Business Suite R12 产品技术白皮书
[3] 用友公司. 用友 ERP 产品方案
[4] 诸学宁. 产品生命周期管理. 北京：中国财政经济出版社，2007
[5] 宋立荣. 供货商的竞争环境及策略分析. 北京：中国流通经济，2001

第 16 章　工作流技术

工作流技术是实现业务过程自动化的关键技术，实行工作流管理能够更加有效地管理各种业务过程。工作流是指业务过程的自动执行，它根据业务过程定义相应的规则，保证文档、信息或任务在业务执行者之间传递，以此帮助业务过程的顺利执行。工作流系统是一个完整的支持业务过程建模、过程执行和管理的软件系统，它根据业务过程的模型来引导过程的执行。

16.1　工作流概述

根据工作流管理联盟（Workflow Management Coalition，WFMC）的定义，工作流就是自动运作的业务过程部分或整体，表现为参与者对文件、信息或任务按照规程采取行动，并令其在参与者之间传递。简单地说，工作流就是一系列相互衔接、自动进行的业务活动或任务。可以将整个业务过程看作是一条河，其中流过的就是工作流。

工作流管理是人与计算机共同工作的自动化协调、控制和通信，在计算机化的业务过程上，通过在网络上运行软件，使所有命令的执行都处于受控状态。在工作流管理下，工作量可以被监督，分派工作到不同的用户达成平衡。工作流管理系统通过软件定义、创建工作流并管理其执行。它运行在一个或多个工作流引擎上，这些引擎解释对过程的定义，与工作流的参与者（包括人或软件）相互作用，并根据需要调用其他的 IT 工具或应用。

总体来说，实际企业中运作的工作流管理系统，是一个人机结合的系统。它的基本功能体现在几个方面：

（1）定义工作流，包括具体的活动、规则等，这些定义是同时被人及电脑所“理解”的。

（2）遵循定义创建和运行实际的工作流。

（3）监察、控制、管理运行中的业务（工作流），例如，任务、工作量与进度的检查、平衡等。

16.1.1　工作流的特征

工作流是一个复杂的系统，它具备自动化、监察和控制、业务重规划等方面的特征。

1．自动化

自动（Automate）是工作流的一个特征，但这主要是指它自动进行的特征，而不是

说没有人的参与。工作流实际上是一个人机协调的混合过程，在一个实际的工作流中，通常总有些步骤是人完成的。协调是工作流管理的一个目标或特征，这包括了人与人、人与计算机、计算机软件之间等多种层面的含义。

2．监察和控制

监察（Monitoring）与控制（Control）是工作流系统的重要功能与特征。这不仅包括对正在发生的业务过程（工作流），还包括它的定义或改变（例如，BPR 的过程）。这是工作流系统带给用户的明显好处之一。

3．业务重规划

从逻辑上，对工作流的关注和研究可以看作是对 BPR 的一种深化。BPR 的观点，要求用户将眼光投向实际业务进行的过程，但这个过程应当是什么样的，怎样分析、构造？工作流就是一个具体的、操作性的答案，它可以令用户从神秘的、难以预测和控制的头脑风暴式的、艺术的业务过程创造，变成解析的、技术的、可控制和预测的工程化过程，如此，才真正体现出“再工程”中“工程”的意义。

工作流与 BPR 的概念，已经被几乎所有的研究者联系在一起研究和应用。在这个领域有一个非常活跃的组织，即国际工作流与再工程协会（Workflow and Reengineering International Association，WARIA）。

无论是从理论和方法上，还是从对象和内容上，都有理由将工作流看作是企业工程的一部分。实际上，已有的关于工作流体系的描述，本身就是一个通用的业务模型框架。仅仅囿于工作流是不够的，必须对整个体系的目标及所有相关要素综合考虑，而这正是企业工程。

16.1.2 工作流的应用现状

工作流技术的发展，经过 20 年的努力，取得了一定的成果。但在实际应用中，应用的企业还是较少，应用的范围窄，效果不理想。这与产品的全面性、集成性有关，也与企业应用的状况有关，具体有以下几点：

（1）工作流底层环境的建立。工作流管理系统必须建立在底层通信的基础上，才能实现分布计算，这要额外付出经费和精力进行构建，这是企业所不希望的，从而限制了在企业中的广泛应用。

（2）标准化程度差。不同的厂商所提供的工作流产品具有独立的一套工作流模型、工作流定义语言、API 函数。但难以在不同系统之间进行交互，集成的效果不理想，不能方便地在不同系统间进行应用对象的重复利用和数据的方便交流。

（3）系统的集成性不理想。工作流管理系统没有达到真正的集成，还是在自己独立地运行，处理一些行政上简单的流程业务，不能很好将 OA、ERP、CRM、SCM 等具体事务处理系统紧密地联系在一起，达到数据录入一次即可有效的目的。

（4）实现的复杂性。实施一个完整的工作流管理系统，是一个复杂的过程，要了解

其他应用系统的业务，进行流程处理分析、业务流程改造、管理规程和操作规范建立等，并且还必须有不同软件供应商的全力配合。

（5）系统的安全性不可靠。系统中对于并发访问和异常错误缺乏正确和可靠的支持。一旦系统出现非正常退出，如何恢复数据并保证数据的一致性还需要解决。

（6）企业管理者的观念改变。一个系统实施的好与坏，与企业的管理者有密切的关系。任何一个应用系统实施成功标记就是：成功=使用。

目前，工作流技术的研究日益受到重视，研究的主要问题包括三个方面：工作流的理论基础，包括体系、模型、语言、接口等；工作流实现技术，包括性能、可靠性方面的研究；工作流技术的应用，包括实施方法、应用集成等。

16.1.3　工作流与传统管理软件

传统的管理软件注重解决企业应用层现存的问题，例如，Excel 可以提高员工画表格的效率；财务软件可以规范财务人员的工作并提高账目查询的效率；CRM 可以规范客户管理，从而使客户资源掌握在公司手中，而不是被一部分业务人员把持，并提高客户响应时间；ERP 解决的是如何配置企业资源，使企业的人力资源、财力资源和物资资源能够根据业务的需求实现最大化配置。

工作流系统关注的是如何缩短流程闲置时间，从而提高企业的业务处理能力，并使企业能够关注真正对企业有意义的增值业务上。

传统软件不能解决工作流的问题，例如，ERP 关注的是企业的资源配置，但不可能解决资源传输过程中的损耗和降低传输（流程）的成本；同样，工作流系统也不能完全解决传统管理软件所能解决的问题，例如，对生产管理的 MRP 系统所能解决的生产过程控制通过工作流系统却很难实现。

一个好的传统软件如果希望能自动化地在整个企业中应用起来，必须有一个强大的逻辑层，用于解决信息传递的逻辑判断和自动流转，这个时候就需要工作流系统的平台。所以，工作流系统比传统管理软件有明显不同：

（1）工作流系统和传统管理软件不是同一种软件，不具可比性。

（2）工作流系统对于已经有传统管理软件的企业的作用非常明显，可以借此平台整合企业的各种应用系统，使之成为一个完整的企业级应用，也就是通常所说的 EAI。

（3）具备工作流系统功能的管理软件（工作流系统与传统管理软件的结合）对于传统管理软件有绝对的优势。

（4）工作流系统可以根据企业的需要，开发解决信息传递问题的流程，帮助企业开发与现有应用系统的接口。

与以往已经被采用的企业 IT 应用体系（如 MRPII、ERP 等）相比，工作流管理系统是一个相当重要的里程碑。从用户的角度来看，工作流管理系统带来（或将要带来）的变化是极其强烈的，甚至可以形容为一种用户梦想的实现。

在一些老的模块化的产品中，系统的设计通常是基于任务分割的，作业项目之间是分裂的。面向对象的技术，并不能直接解决这个的问题，相反，往往使系统变得更加混乱和琐碎。从操作上，典型地，必须不断地在层次结构的功能表（如下拉菜单）或对象之间“进进退退”，或在“神出鬼没”的对象以及相关菜单中捉迷藏。

工作流管理系统是一个真正的人机系统，用户是系统中的基本角色，是直接的任务分派对象，用户可以直接看到计算机针对自己列出的任务清单，跟踪每一项任务的状态，或继续一项任务，而不必从一个模块退出，进入另一个模块，搜索相应任务的线索。也就是说，传统的管理系统是面向功能或对象的，而工作流管理系统是直接面向用户的。这样，用户的任务分派和任务的完成状态，可以被最大程度地计算机化和受到控制。

16.1.4 工作流与 BPR

作为企业流程自动化的应用平台，工作流管理系统最直接的用途就是和企业 BPR 技术相结合，管理企业的各种流程，实现企业流程的自动化。BPR 是对企业过程中的核心流程进行根本的重思考和彻底的重设计，以便在现有衡量企业表现的关键（如成本、品质、服务和速度等）方面获得戏剧化的改善。BPR 是一个很大的概念，国内外研究 BPR 的专家学者很多，也有专门介绍 BPR 的著述。在此，只分析企业使用信息系统后所带来的业务处理流程上的变化，说明企业在使用信息系统后进行必要的 BPR 的必要性。此处所讲的 BPR 仅指在企业使用信息系统后要做的局部流程重组，而不涉及企业商业模式、业务策略的变更等重大业务流程重组。

许多企业对其流程进行了重组，取得了巨大的效果，例如，IBM 信贷公司通过实施 BPR，把为顾客提供融资服务的周期减少了 90%（由原来的 7 天压缩为 4 个小时）；柯达公司对新产品开发实施 BPR，结果把 35mm 焦距一次性照相机从概念设计到生产所需要的开发时间缩短了 50%，从原来的 38 周降低到 19 周。再举一个销售环节的例子。在企业没有使用信息系统时，销售的流程一般是：客户向销售经理下订单并交付预付款（直接给财务部门或由销售经理代收，财务做收入确认），销售经理把订单和预付款收据提交到商务部开提货单，又凭提货单到仓管部门出货并发运，销售经理收到客户的应收账款后，到财务部门给客户开发票。在企业有了信息系统之后，这个处理流程就有了优化的可能，客户基本资料都事先录入到数据库中，销售经理跟客户签订合同以后，及时把合同信息及付款情况输入数据库。客户要求提货时，仓管系统可以根据数据库中未提货的销售合同和客户的资金情况，直接生成提货单，给予提货处理。财务部门同时可给出客户收款情况，进行收入确认，形成应收账款并打印发票；当财务部门收到客户款项时，输入到数据库中，并对用户资金情况进行调整。

类似的例子还有很多，不一而足。严格地说，此处所说的 BPR 不是全盘否定和推翻企业现有的业务流程，而是在有信息系统支撑的假设下，优化现有流程。应该说，实施工作流管理系统的同时，也是企业优化现有业务流程的一次机会，企业可以利用这次机

会，对现行业务流程进行全面盘点和全方位的评估和审视。

工作流管理系统提供了流程自动执行、流程统计分析、实例实时监控和跟踪等功能的一系列软件工具集，一方面实现了流程在计算机上的自动处理，大大缩短了流程的生命周期，提高了企业的工作和生产效率；另一方面，又可以使用户方便地分析企业业务流程，找出不合理之处，快速给出 BPR 的方案。因此，工作流是 BPR 技术的实现和延伸。

由于企业信息化过程是一个循序渐进的过程，导致企业存在许多老的应用系统。加上企业常常根据自己的需要来选择适合自己的应用系统，企业间应用系统的差别更是巨大，企业内部和企业之间各个应用系统不能进行有效的信息交换，企业内部和企业间存在许多信息孤岛。为了消除孤岛，人们提出了许多信息集成框架，例如，基于 XML 的信息集成框架、基于 STEP 标准的工程信息集成框架等，纵观这些技术，笔者认为，它们多局限于静态信息的交换格式的定义，而对于各个应用系统间相互协作、共同完成某项任务的情形却考虑较少。这种情况下，需要多个应用系统按照结构化或非结构化流程来协同工作，在任务的不同时间激活不同的应用系统，并为应用系统传递相应的参数，而工作流管理系统正满足了这一要求。

工作流管理系统可以按照流程的定义，在适当的时间激活相应的应用系统，传递给应用系统相应的参数，获取应用系统的处理结果，把其传递到下一应用，从而实现应用系统的集成。

但是，两者是存在很大区别的。工作流自动化纯粹是软件业提供将工作流程自动化解决方案的范畴。企业 BPR 是分析企业的业务流程，并为了某些方面提高的目标对流程进行修改的行为。

任何组织都可以无须进行 BPR 而通过工作流自动化软件将业务流程自动化。同样，也无须通过工作流自动化而进行 BPR。当然，企业也可以进行 BPR 并导入工作流自动化软件，以使 BPR 的效果更为明显。但没有理由认为两者必须相辅相成或是同一个东西。这一错误认识是由于工作流自动化的概念接受度较慢引起的。工作流自动化是提升企业生产力的解决方案（工具），而 BPR 的概念是改变企业现有工作方式，而改变往往包含了恐惧、不确定性、政策因素和反抗情绪。当工作流自动化被或明确或暗示地等同于 BPR 的时候，这些恐惧、不确定性、政策因素和反抗情绪阻碍了概念的传播。

希赛教育专家提示：当在企业中成功地示范了工作流自动化的好处后，再进行 BPR 将会简单得多。

16.2　工作流系统的实现

工作流传统的实现起始于工作流过程的建模，在完成了过程建模之后，所生成的工作流模型将由工作流执行服务进行实例创建并控制其执行过程。

16.2.1　过程建模

工作流过程模型包含了工作流执行服务运行该过程的所有必需的信息，包括它启动和结束的条件、组成的活动、活动间导航的准则、参与其中的用户、需要激活的应用程序的指针、需要用到的工作流相关数据的定义等。

工作流的建模需要参考组织/角色模型来获得有关组织结构和组织内角色的信息。过程定义指定完成某项活动的组织实体或角色，而不是定义具体人员。工作流执行服务负责在工作流运行环境内将组织实体或角色映射为特定的人员。

工作流建模工具主要用于分析、建模、描述并记录业务过程，它应输出一个能被工作流机动态解释的过程定义。用来进行工作流模型定义和描述的方法有很多，目前较为广泛接受的建模语言有业务过程建模语言（Business Process Modeling Language，BPML）、WFMC 的工作流描述语言、业务过程执行语言（Business Process Execution Language，BPEL）、Petri 网以及 UML 的活动图。

1．业务流程建模语言

BPML 是业务流程建模的元语言，就像 XML 是业务数据建模的元语言一样。它将业务流程描述成控制流、数据流和事件流的结合，在此基础上，还可以在业务流程中添加业务规则、安全规则和事务管理等特性。与传统的流程建模语言相比，它具有描述端到端（end-to-end）流程的能力。这样，可以从多个参与者的角度来审视流程模型。BPML 为协同事务性业务流程提供了一个虚拟执行模型，它基于一个叫做事务有限状态机的概念，所以在一致性检查、防止死锁、瓶颈检测和流程优化方面有较强的能力。BPML 具有如下特征：

（1）BPML 进程跨越多种应用程序和企业，不受任何防火墙限制。

（2）BPML 提供了中间件的特性，它可以独立开发，实现远程处理监控、发布和订阅和消息队列等功能。

（3）BPML 能够处理不同的应用程序，如数据库管理系统、软件构件等。

（4）BPML 能够用进程定义业务事务和系统事务，业务事务经常包括两个以上的参与者，而系统事务能够包括多个应用程序。

（5）统一业务流程和技术是 BPML 的关键目标。

2．WFMC 的工作流描述语言

WFMC 针对工作流建模做了两方面的工作：

（1）建立了一个元模型（Process Meta Model，PMM），它用于描述一个过程模型内各个对象、及它们之间的关系和它们的属性，有利于多个工作流产品之间交换模型信息。WFMC 定义的过程元模型如图 16-1 所示。

（2）定义了一套可以在工作流管理系统之间，以及在管理系统与建模工具之间交互过程模型定义的 API 接口。

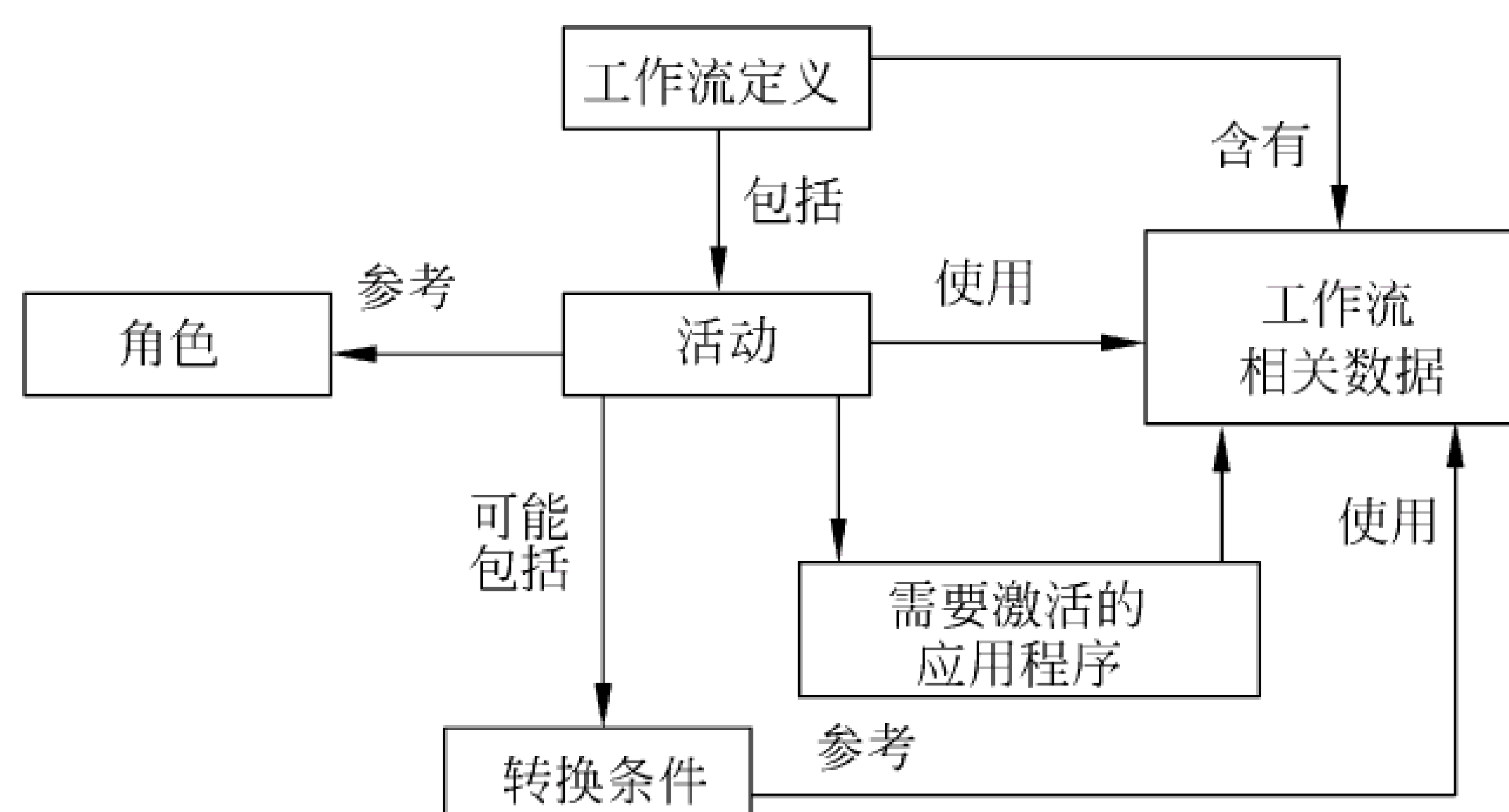


图 16-1　WFMC 定义的过程元模型

3．业务过程执行语言

BPEL 是基于 XML 的语言，设计目标是为多个企业之间使用网络服务的组合，实现分布计算或网格计算环境激活共享任务。BPEL 结合了并且替换了 IBM 的 Web 服务流语言（Web Services Flow Language，WSFL）和 Microsoft 公司的 XLANG 规范。使用 BPEL，程序员在形式上描述了一种发生在网络上的业务处理，其方式为任何互操作的实体都可以按照同样的方式执行一个或多个处理中的步骤。

16.2.2　工作流运行控制

工作流执行服务对使用工作流模型描述的过程进行初始化、调度并监控过程中每个活动的执行，在需要人工介入的场合，完成计算机应用软件与操作人员的交互。实现这个操作的核心是工作流引擎（WorkFlow Engine，WE）。工作流引擎是指工作流作为应用系统的一部分，并为之提供对各应用系统有决定作用的根据角色、分工和条件的不同决定信息传递路由、内容等级等核心解决方案。工作流引擎要完成过程的创建、删除、活动的执行与控制，以及与应用软件和操作人员的交互。

开发一个系统最关键的部分不是系统的界面，也不是和数据库之间的信息交换，而是如何根据业务逻辑开发出符合实际需要的程序逻辑并确保其稳定性、易维护性（模块化和结构化）和弹性（容易根据实际业务逻辑的变化作出程序上的变动，如决策权的改变、组织结构的变动和由于业务方向的变化产生的全新业务逻辑等），工作流引擎解决的就是这个问题。如果应用程序缺乏强大的逻辑层，势必变得容易出错（信息的路由错误、死循环等）。就好比一辆汽车，外表做得再漂亮，如果发动机有问题就只是一个摆设。应用系统的弹性就好比引擎转速方面的性能，加速到 100km 需要 1 个小时（业务流程发生变动，需要进行半年的程序修改）还能叫好车吗？引擎动不动就熄火（程序因为逻辑的问题陷入死循环）的车还敢开吗？

当今社会分工越来越细，在一个单位内部也越来越强调专业化，大部分工作都需要

多个部门和员工合作完成。一个制度良好的单位往往对各种工作的工作流程以文件的形式固定下来，即使是管理不太正规的单位也有约定俗成的工作步骤。这种工作流程保证了一件任务能按预定的顺序从起点流向终点，并且在需要的时候可以跟踪、查询和统计。

要认识工作流引擎，需要了解以下几个要素。

（1）实体（Entity）：是工作流的主体，是需要随着工作流一起流动的物件。例如，在一个采购申请批准流程中，实体就是采购申请单；在公文审批流程中，实体就是公文。

（2）参与者（Participant）：是各个处理步骤中的责任人，可能是人，也可能是某个职能部门，还可能是某个自动化的设备。

（3）流程定义（Flow Definition，FD）：是预定义的工作步骤，它规定了实体流动的路线。它可能是完全定义的，即对每种可能的情况都能完全确定下一个参与者；也可能是不完全定义的，需要参与者根据情况决定下一个参与者。

（4）工作流引擎：是驱动实体按流程定义从一个参与者流向下一个参与者的机制。

可以看出，前三个要素是静态的，而第四个要素是动态的，它将前三者结合起来，是工作流的核心组成元素。

16.2.3　工作流管理中的人机交互

在整个工作流执行中，不同的操作人员需要完成的工作大约可分为以下几种。

1．模型定义

工作流管理软件在企业中要成功应用，最基础、最重要的工作就是业务模型定义，企业的业务过程模型的背后是企业的实际业务流程和各项管控制度。所以，企业业务过程模型定义是企业业务管理专家（非常熟悉企业业务和管控流程的人才）准确提出需求，就其实际而言，这件事就是把企业的业务流程和管控制度抽象、整合优化成一个状态转移矩阵或有限自动状态机。创建、修改和发布企业的业务过程模型，一般由企业的业务管理部门的人员按照企业业务流程完成。

希赛教育专家提示：企业业务过程模型定义过程并不是简单地把企业现有业务流程和管控制度做一个翻译，而是一个流程优化和创新过程，也就是 BPR。

企业业务流程设计最好是使用图形化的流程设计工具，用户可以在 Web 网页上进行流程设计，建立具有复杂分支条件的流转过程，满足各种审批、会签、业务处理、公文流转的需要。

如果企业进行了业务流程的优化，那么业务模型定义需要慎重考虑并广泛评审。

2．人机交互

工作流的人机交互非常重要，因为在工作流管理中，每一个操作者是一个节点，在任何一个节点上出现错误或耽误，工作流就出现滞塞，下一个节点就收不到处理的信息，导致工作很难进行。对工作流管理系统的用户来讲，除流程发起人外，其余人都是被动的任务接受和处理者，他们要按照工作流任务管理器提供的任务项，完成具体的业务处

理工作。参与工作流管理系统的用户涉及面很广，而环节又很多，环节之间的关联度很大，有一个用户掉链子则导致整个流程不畅通，处理不好会导致内部事故，所以，在人机交互设计上，除了遵守一般软件人机交互设计的原则外，还要特别注意以下几点：

（1）按权限设计界面。首先要注意的就是不同权限的人界面不同，最好是个性化的，并和用户的职位和工作相适应。

（2）个性化导航。和传统的菜单式相比，人机交互界面要根据用户的工作类型和工作流程做一个导航，最好是站在用户的角度，按用户日常工作事情的主次或流程排列，即所谓的“角色驱动，流程导航”。

（3）简洁明了。用户在进入工作流管理系统后，一眼就看到系统推给他的审批作业，只需要双击点开，简单批语和点击按钮，就可以顺利实现审批。如果工作流评审推出界面设计不好，可能导致不熟练的用户丢三落四或找不到地方。

（4）多途径提醒方式。系统把审批作业推给下一环节的用户时，至少要有两种以上的通知和提醒方式，例如，工作流系统的提醒外加邮件提醒，还可以有短信提醒等。另外，如果工作流在某一环节滞留的时间超过既定范围，则系统自动产生再次催促，这是非常必要的功能。

（5）支持远程或委托。人机交互界面最好简单，不需要太多的美化装饰，以方便远程评审。在很多情况下，需要在异地或带宽限制的环境下，快捷方便实现审批。还有就是最好能支持短信和手机模式，尤其是 3G 已经出炉的今天。

（6）方便弃审。一旦某用户不小心做了错误的处理，并提交到下一流程，应该能够用弃审的功能恢复审核前的状态，并重新审核。

3．系统运行状态监控

随着计算机网络技术和社会信息化建设的飞速发展，IT 已经成为各个单位提升工作效率，加强自身核心能力的一个不可或缺的重要手段。但信息系统的运行维护正变得越来越难管理，运行维护人员经常被大量的 IT 系统问题困扰。任何组织，只要它的人员和信息系统发展到一定的规模，就不可避免地会面临信息系统管理方面的难题。

所以，系统运行状态监控功能是工作流管理系统的又一个关键功能，没有经验的工作流系统设计者在企业的正常流程上狠下工夫，以为把正常流程设计完了就功德圆满了，而实际的情景是，异常的分支实在是太多了，通常处理异常的工作量可能是处理正常流程工作量的数倍或数十倍，而流程走入一些异常分支，经常需要后台的干预。因此，工作流系统的维护和运行监控工作显得非常重要。系统运行监控的主要工作是检查、监视系统的执行情况，对于系统中出现的意外情况进行紧急处理，例如，终止、恢复某个过程实例的执行，改变某个活动的状态以便整个系统能够继续执行等。

其实，系统运行状态监控是任何机械或信息系统的重要功能，就像小区的物业监控、矿井的安全监控、发电厂的设备监控一样。其功能和作用很容易想象，关键的是，系统

设计者要设计全面、快捷、易于掌握的系统运行监控系统。

16.3 工作流与 ERP

工作流在 ERP 系统的发展中，是一个相当重要的里程碑，对企业 IT 的应用带来的变化是极其强烈的。以工作流为基础可配置可重构的 ERP 系统，可以将工作流和 ERP 的事务处理结合在一起进行考虑，将具有更好的集成性，具有更长的生命周期。国家 863 计划提出的新一代 ERP 系统的标准中，明确要求以工作流引擎为基础，加强流程控制与事务处理系统的集成。

16.3.1 实现 ERP 和 OA 集成

ERP 系统是对企业能够提供业务数据支持的信息系统，OA 系统是实现公文收发、流转、签发、归档等群组化办公作业自动化的信息系统，两者都是为实现单一目标而运行的信息系统。

在企业的业务活动中，经常有些业务是贯穿 ERP 和 OA 两个系统的。例如，在采购流程中，采购申请生成、采购订单生成、验收单生成是在 ERP 系统中进行；采购单审批、入库准备单流转在 OA 系统中进行。企业中存在对 OA 和 ERP 两个系统集成的需求。另外，ERP 系统和 OA 系统实施的难度差别造成一个时期内系统覆盖范围不同，将两个系统集成，ERP 的实施效果可以事半功倍。

将 ERP 和 OA 两个系统集成，涉及到组织、角色、任务和过程的定义和管理。通过工作流系统进行集成，不但可以把两个系统中的多个模型统一，还可以使企业专注于应用业务，更方便地进行 BPR。

对 ERP 和 OA 两个系统的集成，主要的工作有集成方案的确定、系统集成功能范围的确定、工作流系统的创建或改造、组织模型的统一等。

16.3.2 集成方案介绍

针对 ERP 和 OA 系统集成的方案主要有以下三种。

1．更换原有的 ERP 系统

更换原有的 ERP 系统，选择能够同时提供 OA 和 ERP 解决方案的供应商。同时提供 OA 和 ERP 解决方案的供应商，其产品在设计阶段就考虑到了两个系统的集成。但是，目前这样的方案往往是供应商出于市场份额的考虑而提供的，由于开发规模、成本和周期的限制，所提供的 ERP 和 OA 一体化方案的功能往往比较简单，不能满足企业个性化的需求。而且，ERP 系统在企业内运行一段时间后，更换新的系统，会面临新旧系统间数据移植的巨大工作量，用户不愿意舍弃熟悉的界面和高昂的费用等困难。所以，这个

方案只能被未实施 ERP 系统企业中的少部分企业使用。

2．使用 ERP 供应商的合作伙伴提供的集成方案

例如，Lotus Notes 为 SAP、Oracle、JDE 等公司的 ERP 产品都提供了集成化的解决方案。其方法是：在 OA Server 和 ERP Server 之间通过数据库连接工具 DECS 连接。在 ERP 系统的 DB 建立大量视图供 OA 访问，在 OA Server 上建立关系型数据库，存储定期从 ERP 系统中按照字段映射过来的静态数据，作为 OA 系统组织和资源定义的依据。OA 系统中的表单鉴审后可以通过 ERP 系统的 Interface table 写入 ERP 系统。

这种方案可以实现两个系统的高度集成，但是存在以下问题。

（1）不是所有的 ERP 系统都有相应的集成方案提供。Lotus Notes 仅对大型而且著名的 ERP 产品提供了这样的集成方案。

（2）这个方案的实现和维护费用非常高。如果是新增流程，需要在 ERP 系统中新增视图，在 OA 系统中新增表单。对于大型的 ERP 系统，其数据库中的表有近万个，加上在 OA 中创建表，都是企业 IT 人员无法独立完成的，仍需要方案提供者的服务。即使是方案的提供者，在使用这种工具完成两个应用系统结合时，也必须同时对两个系统了如指掌。然而，不论在国内还是国外，同时能够深层次了解两个系统的技术人员极为紧缺，加上高昂的购买费用，企业很难接受。

（3）ERP 实施模块增加，特别是 ERP 系统的升级，都会造成集成化系统的瘫痪，限制了企业的业务发展。

因此，此方案的应用仍然比较少。

3．通过工作流系统，实现工作流程在两个平台上切换

在工作流系统的管理下，用户通过远程登录工具和模拟键盘录入，实现 OA 平台和 ERP 平台之间的简单切换。系统架构如图 16-2 所示。

工作流在 ERP 中的应用比较晚，提供工作流管理功能的软件产品也不多，典型的有国外的 Oracle、SAP、JDE 等少数几家。相对于独立的工作流软件来说，把自己的 OA 系统和 ERP 系统结合起来，已经出现了不少的产品。例如，用友、金碟等大的 ERP 厂商推出的方案中已经包含了 OA 和 ERP 系统。这些产品仍然是把 OA 和 ERP 系统单独分开，但是他们能够和自己公司的系统进行集成。例如，用友 OA 和用友的 U8、NC 等产品可以集成，但是不能够和 SAP、金蝶的 ERP 集成。国内的产品存在下面几个特点：

（1）系统主要构建在 Sun 的 J2EE 平台和 Microsoft 的.Net 平台上。

（2）应用范围比较窄，尚未形成独立产品。

（3）技术尚未完全成熟。

中国的企业如何加强在流程方面的管理，提高企业的竞争力，已经成为这些企业下一步的目标。伴随着科技的不断发展和企业用户的不断需求，工作流系统将会应用得越来越广泛。

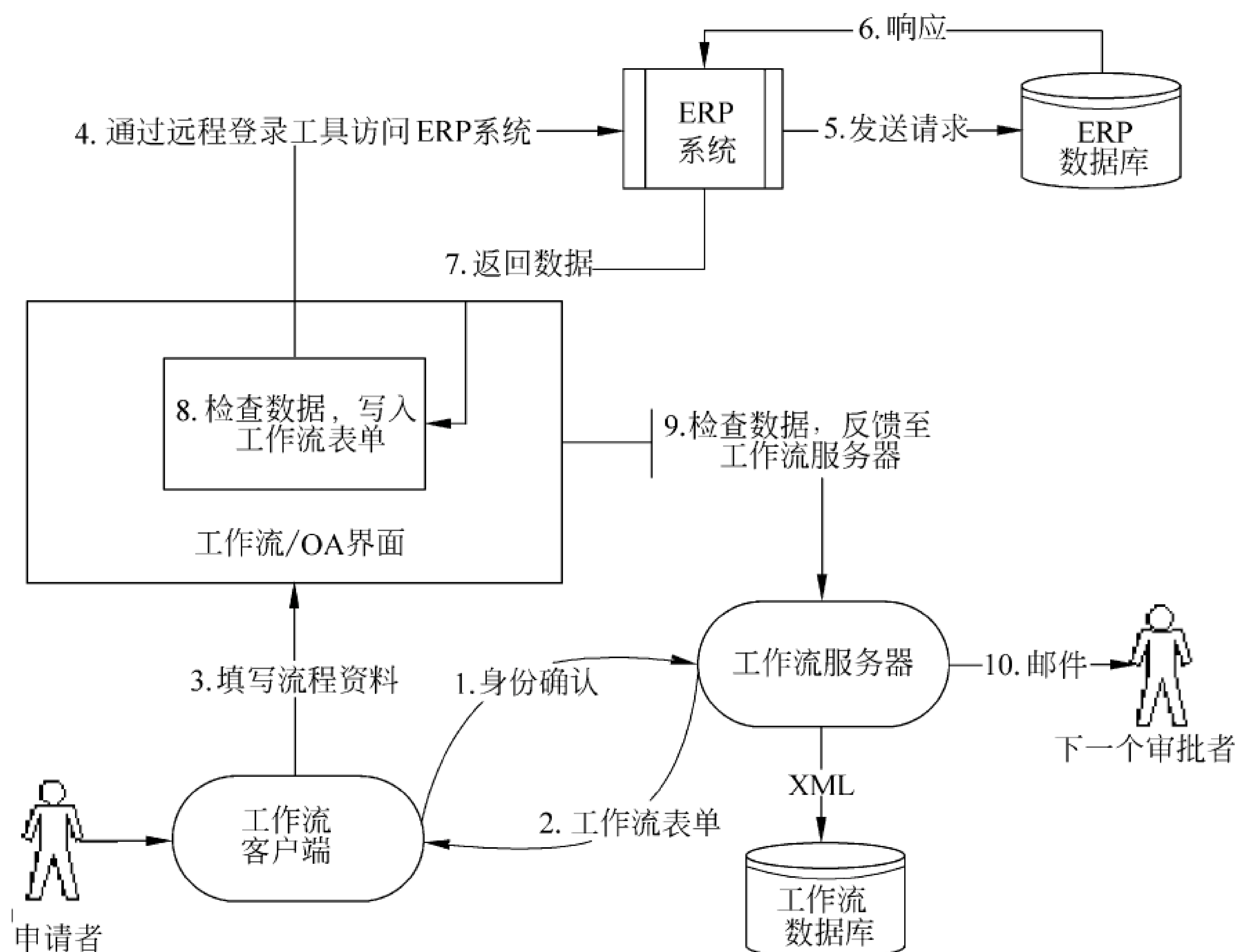


图 16-2 集成后的系统架构

本章参考文献

[1] 蒋明炜，戴宝纯，吴英. 工作流管理与 ERP 的应用. 中国计算机报，2003

[2] 崔永圣. 基于 Web 的工作流管理系统的设计与实现.http://www.e-works.net.cn/ewkArticles/Category175/Article14084.htm

[3] Margie Virdell. 服务世界中的业务流程和工作流. http://www-900.ibm.com/developerWorks/cn/webservices/ws-work/index.shtml

[4] 王建民，闻立杰. 工作流管理——模型方法和系统. 北京：清华大学出版社，2004

[5] 范玉顺. 工作流管理技术基础. 北京：清华大学出版社，2001

第 17 章　软件产品线

软件产品线（Software Product Line，SPL）是一个十分适合专业的软件开发组织的软件开发方法，能有效地提高软件生产率和质量、缩短开发时间、降低总开发成本。软件产品线也是一个新兴的、多学科交叉的研究领域，研究内容和范围都相当广泛。

17.1　软件产品线概述

产品线的起源可以追溯到 1976 年 Parnas 对程序族的研究。软件产品线的实践早在 20 世纪 80 年代中期就出现。最著名的例子是瑞士 CelsiusTech 公司的舰艇防御系统的开发，该公司从 1986 年开始使用软件产品线开发方法，使得整个系统中软件和硬件在总成本中所占比例之比从使用软件产品线方法之前的 65:35 下降到使用后的 20:80，系统开发时间从近 9 年下降到不到 3 年。据 HP 公司 1996 年对 HP、IBM、NEC、AT&T 等几个大型公司的分析研究，他们在采用了软件产品线开发方法后，使产品的开发时间减少 1.5～2 倍，维护成本降低 2～5 倍，软件质量提升 5～10 倍，软件重用达 50%～80%，开发成本降低 12%～15%。

虽然软件工业界已经在大量使用软件产品线开发方法，但是正式的对软件产品线的理论研究到 20 世纪 90 年代中期才出现，并且早期的研究主要以实例分析为主。到了 20 世纪 90 年代后期，软件产品线的研究已经成为软件工程领域最热门的研究领域。得益于丰富的实践和软件工程、软件架构、软件重用技术等坚实的理论基础，对软件产品线的研究发展十分迅速，目前软件产品线的发展已经趋向成熟。

软件产品线方法是软件工程领域中软件架构和软件重用技术发展的结果。目前，软件产品线没有一个统一的定义，常见的定义有：

（1）将利用了产品间公共方面、预期考虑了可变性等设计的产品族称为产品线（Weiss 和 Lai）。

（2）产品线就是由在系统的组成元素和功能方面具有共性（commonality）和个性（variability）的相似的多个系统组成的一个系统族。

（3）软件产品线就是在一个公共的软件资源集合基础上建立起来的，共享同一个特性集合的系统集合（Bass，Clements 和 Kazman）。

（4）一个软件产品线由一个产品线架构、一个可重用构件集合和一个源自共享资源的产品集合组成，是组织一组相关软件产品开发的方式（Jan Bosch，JB）。

相对而言，CMU/SEI 对软件产品线的定义，更能体现软件产品线的特征：“产品线

是一个产品集合，这些产品共享一个公共的、可管理的特征集，这个特征集能满足选定的市场或任务领域的特定需求。这些系统遵循一个预描述的方式，在公共的核心资源（Core Assets，CA）基础上开发的”。

根据 SEI 的定义，软件产品线主要由两部分组成：核心资源、产品集合。核心资源是领域工程的所有结果的集合，是产品线中产品构造的基础。也有人将核心资源库称为“平台（platform）”。核心资源必定包含产品线中所有产品共享的产品线架构，新设计开发的或通过对现有系统的再工程得到的、需要在整个产品线中系统化重用的软件构件。与软件构件相关的测试计划、测试实例以及所有设计文档，需求说明书、领域模型和领域范围的定义也是核心资源。产品线架构和构件是用于软件产品线中的产品构建的核心资源最重要的部分。

软件产品线开发有 4 个基本技术特点：过程驱动、特定领域、技术支持和架构为中心。与其他软件开发方法相比，软件开发组织选择软件产品线的宏观上的原因有：对产品线及其实现所需的专家知识领域的清楚界定，对产品线的长期远景进行了战略性规划。

17.2 软件产品线的过程模型

本节介绍三种主要的软件产品线过程模型，分别是双生命周期模型、三生命周期模型和 SEI 的模型。

17.2.1 双生命周期模型

双生命周期模型是最初的和最简单的软件产品线开发过程，它分成两个重叠的生命周期：领域工程和应用工程。两个周期内部都分成分析、设计和实现三个阶段，如图 17-1 所示。

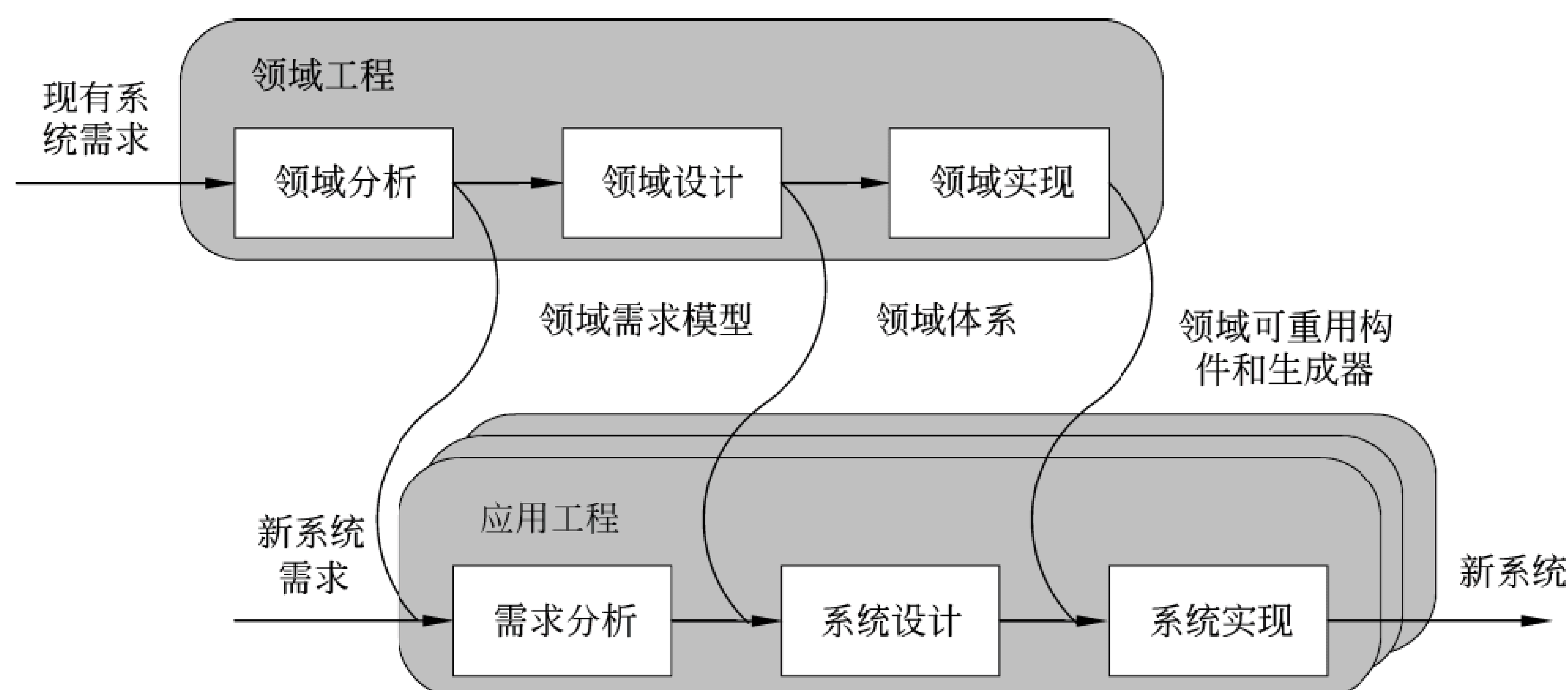


图 17-1 产品线的双生命周期模型

领域工程阶段的主要任务如下。

（1）领域分析：利用现有系统的设计、架构和需求建立领域模型。

（2）领域设计：用领域模型确定领域/产品线的共性和可变性，为产品线设计架构。

（3）领域实现：基于领域架构开发领域可重用资源（构件、文档、代码生成器）。

应用工程在领域工程的基础上构建新产品。应用工程需要根据每个应用独特的需求，经过以下阶段，生成新产品。

（1）需求分析：将系统需求与领域需求比较，划分成领域公共需求和独特需求两部分，得出系统说明书。

（2）系统设计：在领域架构基础上，结合系统独特需求，设计应用的软件架构。

（3）系统实现：遵照应用架构，用领域可重用资源实现领域公共需求，用定制开发的构件满足系统独特需求，构建新的系统。

应用工程将产品线资源不能满足的需求返回给领域工程，以检验是否将之合并入产品线的需求中。领域工程从应用工程中获得反馈或结合新产品的需求，进入又一次周期性发展，称之为产品线的演化。

双生命周期模型定义了典型的产品线开发过程的基本活动、各活动的内容和结果，以及产品线的演化方式。这种产品线方法综合了软件架构和软件重用的概念，在模型中定义了一个软件工程化的开发过程，目的是提高软件生产率、可靠性和质量，降低开发成本，缩短开发时间。

17.2.2　SEI 模型

SEI 将产品线的基本活动分为三部分，分别是核心资源开发（即领域工程）、产品开发（即应用工程）和管理。主要特点如下。

（1）循环重复是产品线开发过程的特征，也是核心资源开发、产品线开发以及核心资源和产品之间协作的特征。

（2）核心资源开发和产品开发没有先后之分。

（3）管理活动协调整个产品线开发过程的各个活动，对产品线的成败负责。

（4）核心资源开发和产品开发是两个互动的过程，三个活动和整个产品线开发之间也是双向互动的。

有关 SEI 模型的更加详细的知识，将在 17.6 节进行介绍。

17.2.3　三生命周期模型

美国南加洲大学的 Frederick P. Brooks 博士在针对大型软件企业的软件产品线开发对双生命周期模型进行了改进，提出了三生命周期软件工程模型，如图 17-2 所示。

三生命周期模型为有多个产品线的大型企业增加企业工程流程，以便在企业范围内对所有资源的创建、设计和重用提供合理规划。为了强调产品线工程在满足市场需求上

与一般的系统重用的区别，在领域工程中增加了产品线确定作为起始阶段，和领域分析阶段、架构开发阶段、基础资源开发阶段组成整个领域工程，还为领域分析阶段增加市场分析的任务；同样，为应用领域增加了业务市场分析与计划。在领域工程和应用工程之间的双向交互中添加核心资源管理作为桥梁，核心资源管理和领域工程、应用工程之间的支持和交互是双向的，以便于产品线核心资源的管理和演化。

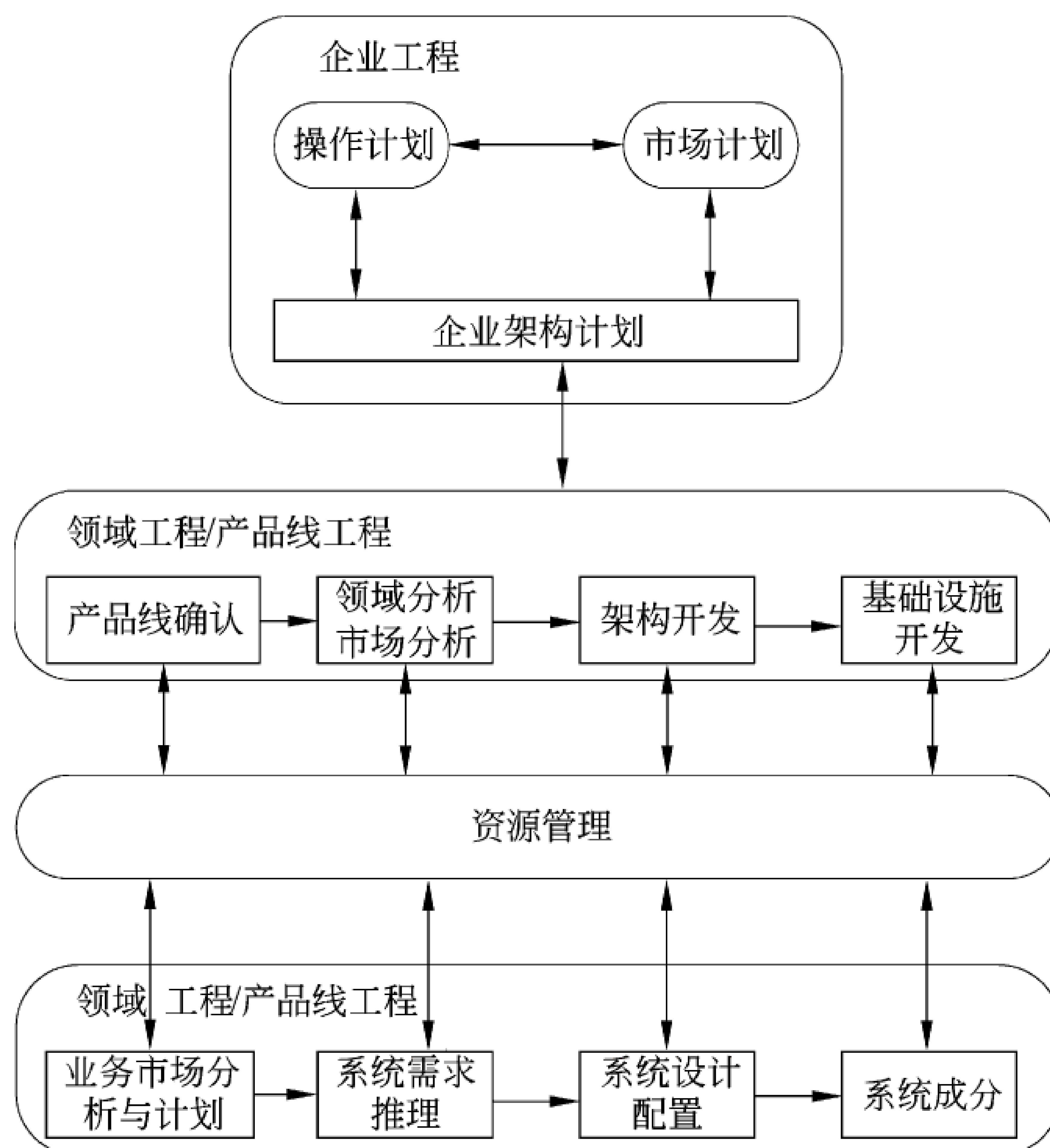


图 17-2 产品线的三生命周期模型

以上描述的软件产品线开发过程并没有明确描述如何重用软件组织内遗留资源（Legacy Assets，LA）。实际上，大多数将要建立软件产品线的组织都积累有产品线所在领域的大量应用代码和相关文档，这些代码和文档中包含的知识对领域工程来说是至关重要的。Boeing 公司的 Margaret J.Davis 将软件再工程和产品线方法结合，该方法将软件再工程应用于领域工程中，用一种系统化的方法挖掘遗留系统中的知识。根据产品线和遗留系统采用技术的差异大小，能恢复出的资源可能包括人员组织的交互和过程信息、软件架构和高层设计、算法代码和过程等。

17.3 软件产品线的组织结构

软件产品线开发过程分为领域工程和应用工程，相应的软件开发的组织结构也应该

有两个基本组成部分：负责核心资源的小组、负责产品的小组。这也是产品线开发与独立系统开发的主要区别。

基于对产品线开发的认识不同以及开发组织背景不同，有很多组织结构方式。但可以根据是否有独立的负责核心资源开发的小组分为两大类。其中设立独立小组的典型的组织结构如图 17-3 所示。其中架构组监控核心资源开发组和产品开发组以保证核心资源和产品能够遵循架构，同时负责架构的演化。配置管理组维护每个资源的版本。架构组、核心资源开发组与负责独立产品开发的小组互相独立。

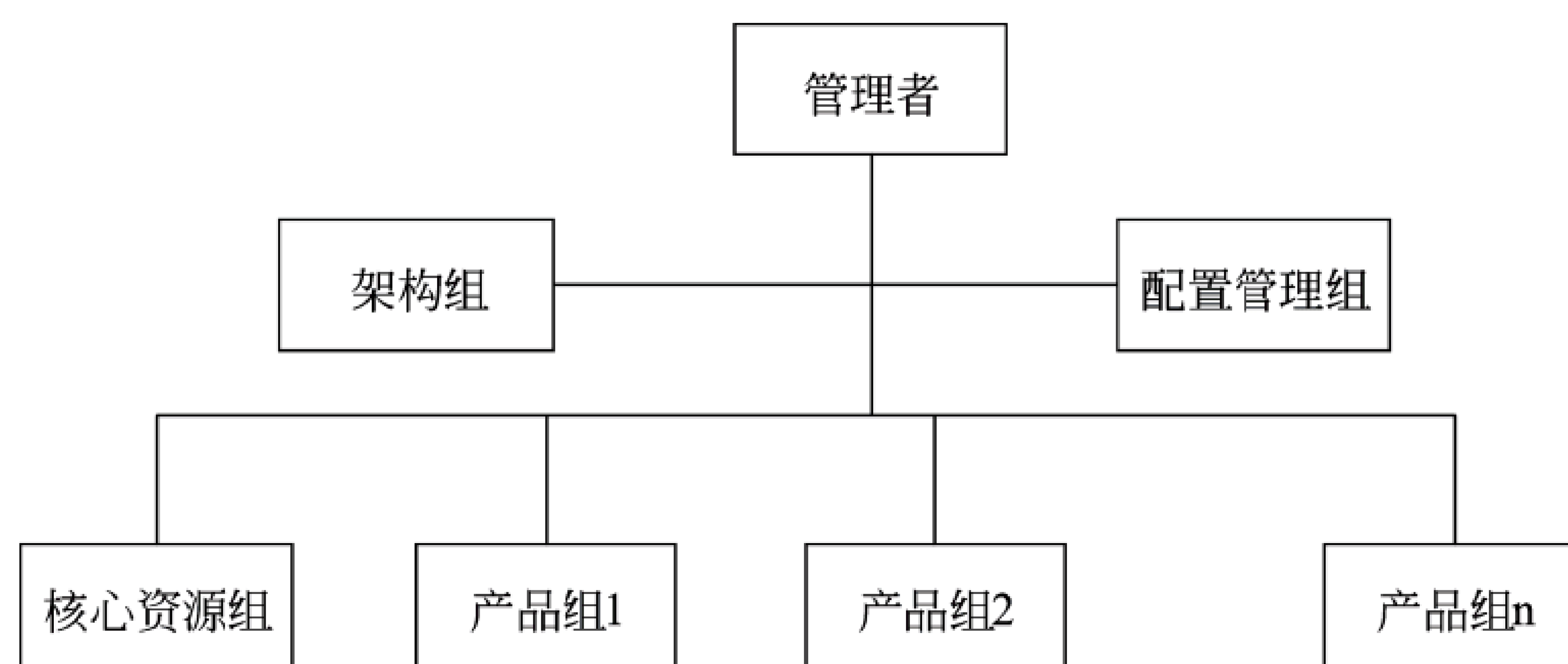


图 17-3　典型产品线开发组织结构

17.3.1　SEI 组织结构

SEI 在其推荐的组织结构中强调市场人员在获取需求和推介产品中的作用。将产品线组织分为 4 个工作小组。

（1）市场人员：产品线和产品能力、客户需求之间的沟通桥梁。

（2）核心资源组：负责架构和其他核心资源的开发。

（3）应用组：负责交付给客户的系统的开发。

（4）管理者：负责开发过程的协调、商务计划等。

SEI 还将客户提出的需求和对系统的反馈作为产品线组织的重要外部组织接口。

设有独立核心资源小组的组织结构通常适合于至少由 50～100 人组成的较大型的软件开发组织，设立独立的核心资源小组可以使小组成员将精力和时间集中在核心资源的设计和开发上，得到更通用的资源。但独立的核心资源小组很容易迷失于建立极好的高度抽象、高度可重用的核心资源上，而忽视了这些资源对应用工程中需求的满足程度，因为这样的结构容易抑制应用工程中的反馈，使得所开发的核心资源无法在整个产品线中获得良好的应用。

另外一种典型的组织结构是不设立独立的核心资源小组，核心资源的开发融入各系统开发小组中，只是设立专人负责核心资源开发的管理。这种组织结构的重点不在核心资源的开发上，所以比较适合于组成产品线的产品共性相对较少，开发独立产品所需的

工作量相对较大的情况。也是小型软件组织向软件产品线开发过渡时采用的一种方法。

17.3.2 组织模型

Jan Bosch 在研究了众多采用软件产品线开发方法的公司后，将软件产品线的组织结构归纳为 4 种组织模型。

（1）开发部门（Development Unit，DU）：所有的软件开发集中在一个部门，每个人都可承担领域工程和应用工程中适当的任务。这种结构比较简单，有利于沟通，适用于不超过 30 人的组织。

（2）业务部门（Business Unit，BU）：每个部门负责产品线中一个或多个相似的系统，共性资源由需要使用它的一个或几个部门协作开发，整个团队都可享用。这种结构资源更容易共享，适用于 30～100 人的组织，主要缺点是业务部门更注重自己的产品而将产品线的整体利益放在第二位。

（3）领域工程部门（Domain Engineering Unit，DEU）：有一个专门的单位（领域工程部门）负责核心资源库的开发和维护，其他业务部门使用这些核心资源来构建产品。这种结构可有效的降低通信的复杂度、保持资源的通用性，适用于超过 100 人的组织。缺点是难以管理领域工程部门和不同产品工程部门之间的需求冲突和因此导致的开发周期延长。

（4）层次领域工程部门（Hierarchical Domain Engineering Unit，HDEU）：对于非常巨大和复杂的产品线，可以设立多层（一般为两层）领域工程部门，不同层部门服务的范围不同。这种模型趋向臃肿，对新需求的响应比较慢。

对于中小型软件开发组织来说，笔者建议采用一种动态的组织结构，根据产品线的建立方式和发展阶段、成熟程度的变化，由一种组织结构向另一种组织结构演变。这种方法的主要依据是在产品线不同发展阶段，领域工程和应用工程的在总工作量中所占的比例是不同的。例如，对于从零开始建立的产品线，在其建立初期，核心资源的开发工作量要大大多于产品的开发。此时集中力量组织成专门的小组进行核心资源的开发，当核心资源基本完成时，可以将该小组部分成员逐步转移到产品开发中。而对于已有多个产品的情况下建立产品线的演变过程使用相反的方向更为合适。

这种动态的组织结构可以使中小型组织采用产品线开发方式造成的在人力资源上的压力得到缓解，使人力资源的需求在产品线的整个开发过程中趋于平稳。人员在两种小组之间的流动可以使流动人员作为小组之间信息交流的一种补充方式，虽然这不是一种最好的、合乎规范的信息交流方式，但毕竟也是一种快速有效的方式。

希赛教育专家提示：组织结构的变化对产品线来说是一个很重要的问题，需要制定相应的变化规划并要有良好的管理技术的支持来保证整个产品线的成功。

17.4 软件产品线的建立方式

软件产品线的建立需要软件组织有意识地、明显地努力才有可能成功。软件产品线

的建立通常有 4 种方式，其划分依据有两个：

（1）该组织是用演化方式（evolutionary）还是革命方式（revolutionary）引入产品线开发过程。

（2）是基于现有产品还是开发全新的产品线。

4 种方式的基本特征如表 17-1 所示。

表 17-1　软件产品线建立方式基本特征

	演 化 方 式	革 命 方 式
基于现有产品集	基于现有产品架构开发产品线的架构；经演化现有构件的文件一次开发一个产品线构件	产品线核心资源的开发基于现有产品集的需求和可预测的、将来需求的超集
全新产品线	产品线核心资源随产品新成员的需求而演化	开发满足所有预期产品线成员的需求的产品线核心资源

1．将现有产品演化为产品线

在基于现有产品架构设计的产品线架构的基础上，将特定产品的构件逐步地、越来越多地转化为产品线的共用构件，从基于产品的方法“慢慢地”转化为基于产品线的软件开发。这种方式的主要优点是通过对投资回报周期的分解、对现有系统演化的维持使产品线方法的实施风险降到了最小，但完成产品线核心资源的总周期和总投资都比使用革命方式要大。

2．用软件产品线替代现有产品集

基本停止现有产品的开发，所有努力直接针对软件产品线的核心资源开发。遗留系统只有在符合架构和构件需求的情况下，才可以和新的构件协作。这种方法的目标是开发一个不受现有产品集存在问题的限制的、全新的平台、总周期和总投资较演化方法要少，但因重要需求的变化导致的初始投资报废的风险加大。另外，基于核心资源的第一个产品面世的时间将会推后。

现有产品集中软硬件结合的紧密程度，以及不同产品在硬件方面的需求的差异，也是产品线开发采用演化还是革命方式的决策依据。对于软硬件结合密切且硬件需求差异大的现有产品集因无法满足产品线方法对软硬件同步的需求，只能采用革命方式替代现有产品集。

3．全新软件产品线的演化

当一个软件组织进入一个全新的领域，要开发该领域的一系列产品时，同样也有演化和革命两种方式。演化方式将每一个新产品的需求与产品线核心资源进行协调。这种方式的好处是先期投资少，风险较小，第一个产品面世时间早。另外，因为是进入一个全新的领域，演化方法可以减少和简化因经验不足造成的初始阶段错误的修正代价；缺

点是已有的产品线核心资源会影响新产品的需求协调，使成本加大。

4．全新软件产品线的开发

架构设计师和工程师首先要得到产品线所有可能的需求，基于这个需求超集来设计和开发产品线核心资源。第一个产品将在产品线核心资源全部完成之后才开始构造。这种方法的优点是一旦产品线核心资源完成后，新产品的开发速度将非常快，总成本也将减少；缺点是对新领域的需求很难做到全面和正确，使得核心资源不能像预期的那样支持新产品的开发。

17.5 框架和应用框架技术

随着软件技术的发展，软件重用已经从模块、对象的重用发展到了基于构件的重用和基于框架（framework）的重用，这也是当前最主要的两个软件重用的方式。从重用粒度上看，框架要比构件大。框架重用是一种面向领域的软件重用方式，更适合于软件产品线。框架一般建立在同一个或相似领域中，即所要开发的软件系统要具有较强的相似性，通过框架把领域中不变或易变的部分在一定时间间隔内固定下来，把易变的部分以用户接口的形式保留下来，从而达到设计和代码的重用。框架技术与构件技术的结合产生了基于构件的应用框架技术，这是框架技术的一个发展趋势。除此之外，本节讨论的框架主要指面向对象领域中的框架。这是因为框架技术与面向对象技术关系十分密切，如框架的基础实现技术“动态绑定”（binding）就是由面向对象语言的多态机制支持的，并且很多具体的框架技术都是在面向对象环境中描述的。

17.5.1 框架的概念

最早的对框架描述由 Deutsch 在 1983 年给出：“多个抽象类和它们相关算法的集合可组成一个框架，该框架在特定应用中可以通过添加专用代码来将具体子类组织在一起”。

其他对框架的比较重要的定义和描述如下。

（1）Johnson 和 Foot：框架是封装了特定应用族抽象设计的抽象类的集合，框架又是一个模板，关键的方法和其他细节在框架实例中实现。

（2）Buschmann：框架是一个可实例化的、部分完成的软件系统或子系统，定义了一组系统或子系统的架构，并提供了构建系统的基本构造模块，还定义了对特殊功能实现所需要的调整方式。在一个面向对象的环境中，框架由抽象类和具体类组成，框架的实例化包括现有类的实例化和衍生。

（3）Johnson：框架=模式+构件。框架是由开发人员定制的应用系统的骨架（skeleton），是整个系统或子系统的可重用设计，由一组抽象构件和构件实例间的交互方式组成。

以上是对一般框架概念的描述，软件产品线中的框架主要指的是应用框架。对应用框架的描述和定义主要有以下一些。

（1）Gamma：应用框架又称为通用应用，是为一个特定应用领域的软件系统提供可重用结构的一组相互协作的类的集合。

（2）Buschmann：特定领域应用的框架被称为应用框架。

（3）Froehlich：应用框架就是某个领域公共问题的骨架式解决方案。框架为该领域所有应用提供公共的架构和功能基础。

（4）Batory：应用框架技术是用于应用产品线的、通用的、面向对象的代码结构化技术。一个框架就是表达抽象设计的抽象类的集合；框架实例就是为可执行子系统提供的抽象类的子集的具体类的集合。框架是为了重用而设计的；抽象类封装了公共代码，具体类封装特定实例的代码。

经过分析，可以得出以上众多对应用框架的描述的共同点：

（1）应用框架解决的是一个领域或产品族的问题，规定了问题应该如何分解。

（2）包含了应用或子系统的设计，由一个互相协作的类或构件集合组成。

（3）可以通过继承或类的组合来创建应用。

对框架技术的基本特征总结如下。

（1）反向控制：类库是客户代码调用库中已存在类的方法。框架内嵌了控制流，框架调用客户代码（加入框架的新构件和抽象类的方法实例）。

（2）可重用性：框架提供了设计和代码的重用能力。

（3）扩展性：为计划的变化提供了“热点”（hotspot）或“钩子”（hook）等显式说明方式。

（4）模块化或构件化：框架有固定的、稳定的接口和封装的热点。

17.5.2　框架的建立方式

一般框架有三种建立方式：自顶向下、自底向上和混合方式。因为应用框架和软件产品线之间的密切关系，前两种框架建立方式与建立全新的软件产品线时的革命方式和演化方式十分类似，也具有相同的过程和优缺点。混合方式指在大型应用框架的建立过程中，先将应用领域划分为不同的子区域，再分别解决，最终集成为一个完整框架的做法。

根据框架的使用和扩展方式，可以将框架分为两大类：黑盒框架和白盒框架。

黑盒框架通过构件/类的组合来支持重用和扩展。应用中的类由框架的不同构件组合而成。在框架所在领域中，每个构件都有一个预定义的标准接口，一组共享相同接口但能满足不同应用需求的构件组成一个“插接兼容”的构件集合。

白盒框架一般使用类的继承机制实现，由未完成的类（抽象类）组成，类有一个或

多个抽象接口或虚方法。应用需要在抽象类的继承子类中提供特定意义的方法实例来重用框架。开发者通过虚方法的实例化将特定应用的代码联入框架来生成应用，所以虚方法又称为“钩子”或“热点”。

白盒重用需要对框架有很好的理解，生成紧耦合系统。黑盒重用不需要对框架的内部结构有太多的了解，产生松耦合系统。具体的框架实际上都是灰色的，是可继承和可组合方式的结合。

灰色框架可以分成三部分：固定的、可选择的和开放的。框架的固定部分包含了该领域最基本的功能、内建了应用的控制流，由框架主干实现，对应着领域共用部分。框架的可选择部分为该领域中相对固定的、应用特定的功能特征即领域个性部分，用可组合的类或构件实现，在构建应用时在这些构件或类中进行选择、组合。对一些无法准确估计和预测的功能特征即框架的开放部分，只能为其规定统一的接口和与框架的挂接点，用可继承的抽象类的方式来实现，这些部分可以根据应用的具体需求变化进行单独的调整。

与架构的层次结构类似，框架也可设计为层次结构，可称之为层次框架。例如，把一个完整的框架划分为应用框架、领域框架、支撑框架多个层次，框架层次间是标准的或统一的接口。层次框架与层次架构具有相同的优点。

17.6 软件产品线基本活动

本节以 SEI 的过程模型为线索，讨论软件产品线开发的基本活动。

17.6.1 过程模型

从本质上看，产品线开发包括核心资源库的开发和使用核心资源的产品开发，这两者都需要技术和组织的管理。核心资源的开发和产品开发可同时进行，也可交叉进行，例如，新产品的构建以核心资源库为基础，或核心资源库可从已存在的系统中抽取。有时，我们把核心资源库的开发也称为领域工程，把产品开发称为应用工程。图 17-4 说明了产品线各基本活动之间的关系。

每个旋转环代表一个基本活动，三个环连接在一起，不停地运动着。三个基本活动交错连接，可以任何次序发生，且是高度重叠。旋转的箭头表示不但核心资源库被用来开发产品，而且已存在的核心资源的修订甚至新的核心资源常常可以来自产品开发。

在核心资源和产品开发之间有一个强的反馈环，当新产品开发时，核心资源库就得到刷新。对核心资源的使用反过来又会促进核心资源的开发活动。另外，核心资源的价值通过使用它们的产品开发来得到体现。

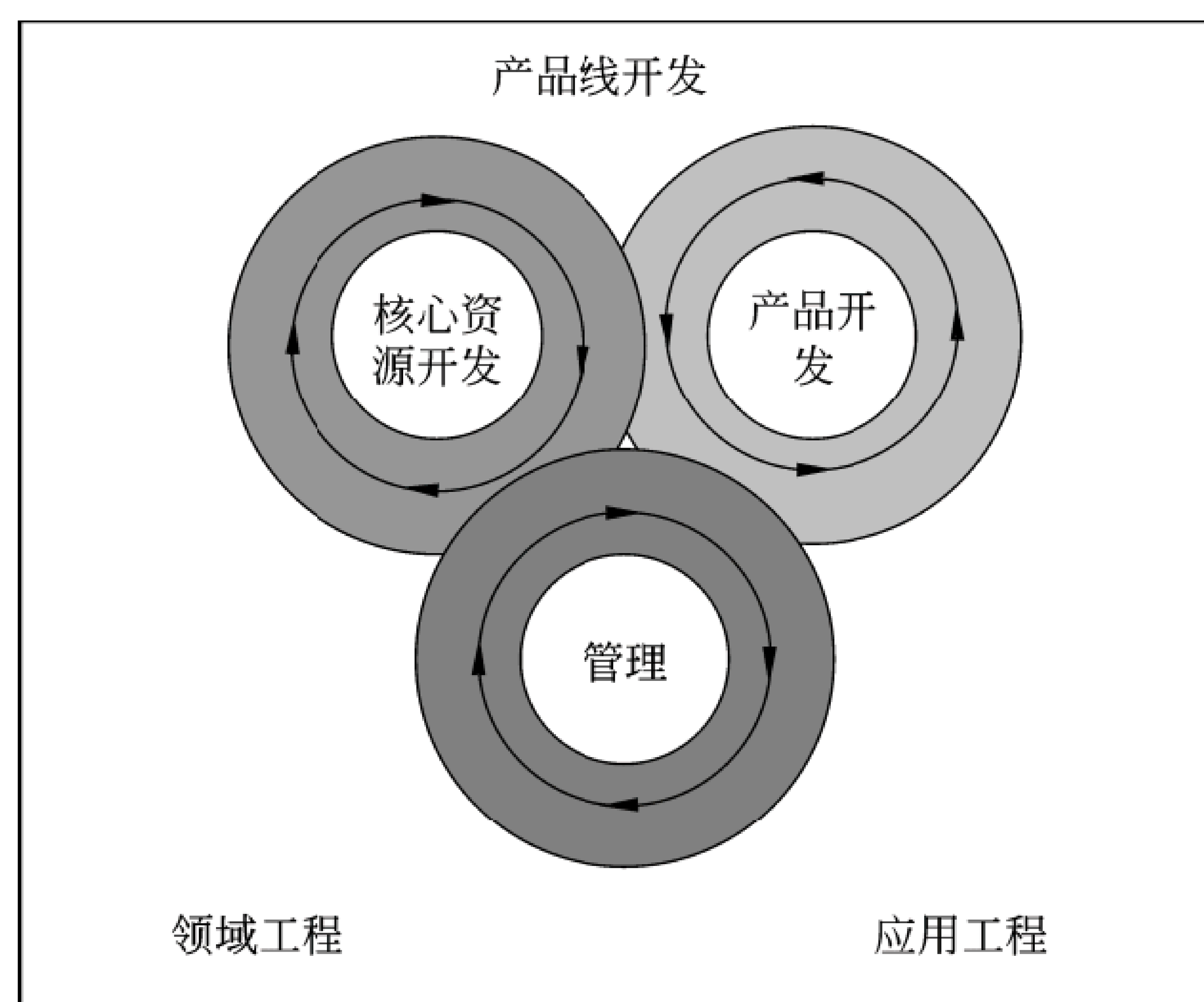


图 17-4　产品线基本活动

17.6.2　产品线分析

产品线分析是产品线的需求工程，是业务机会的确认和产品线架构的设计之间的桥梁。产品线分析强调：

（1）通过捕获项目干系人（风险承担者）的意见来揭示产品线需求。

（2）通过系统的推理和分析、集成功能需求和非功能需求来完成产品线需求。

（3）产品线设计师对产品线需求的可用性。

1．上下文

产品线的开发包括资源开发、产品计划和产品开发几个步骤，产品线分析是资源开发的一部分，如图 17-5 所示。

产品线分析是把对业务机会的初步确认细化为需求模型，对正在开发的产品线而言，捕获：

（1）组织的商业目标和约束。

（2）包含在产品线中的产品。

（3）最终用户和其他项目干系人的需求。

（4）大粒度重用的机会。

分析能否为并行开发提供机会，对产品线开发来说是至关重要的。资源开发需要固定投资，特别是及时的投资，但产品线的成功却往往取决于组织快速进入市场的能力。减少产品线进入市场时间的唯一途径就是使资源开发并行进行。对产品线分析而言，这意味着要尽可能快地发现重大设计信息。

2．项目干系人观点

产品线项目干系人是指人或受产品线开发所影响的系统，一个特定的产品线的干系人可以包括（但不限于）决策者、市场分析员、技术经理、产品线分析员、设计师和程序员、产品分析员、设计师和程序员、产品的最终用户、与产品线中的产品交互的内部和外部系统、政府机构、保险公司等。

每个产品线干系人对产品线都有自己的看法，也就是一组期望和对产品线的需求。因为许多干系人对产品线有同样的期望和需求，因此，只需要关注那些起关键作用的干系人。

对产品线开发来说，关键的干系人包括决策者、最终用户和产品线开发人员，如图17-6所示。决策者把产品线看作是达到组织目标的机制，最终用户注重产品线中的特定产品所能提供的服务，产品线开发人员注重架构、产品计划和生产产品线中的产品所需的构件。

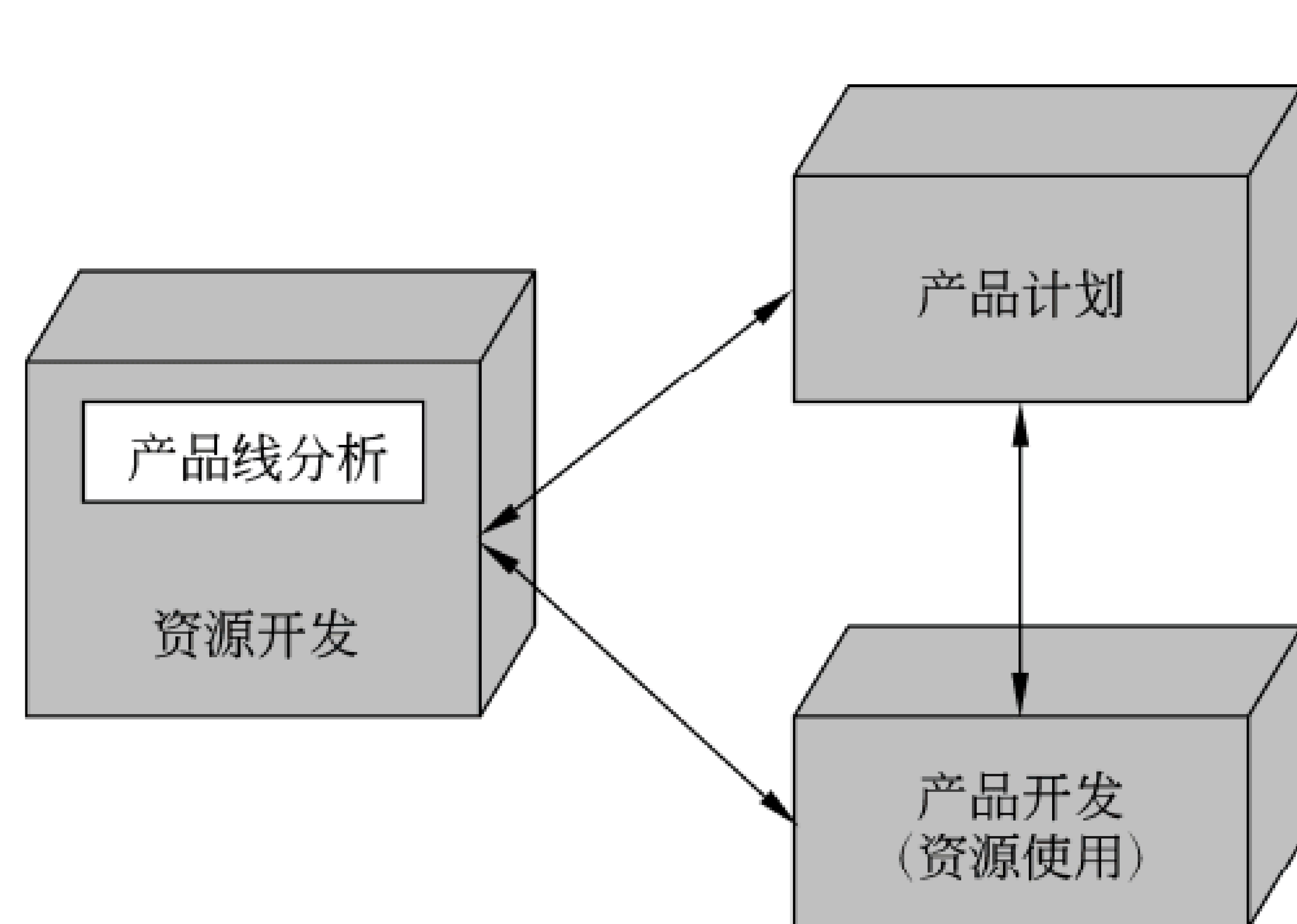


图17-5　产品线分析

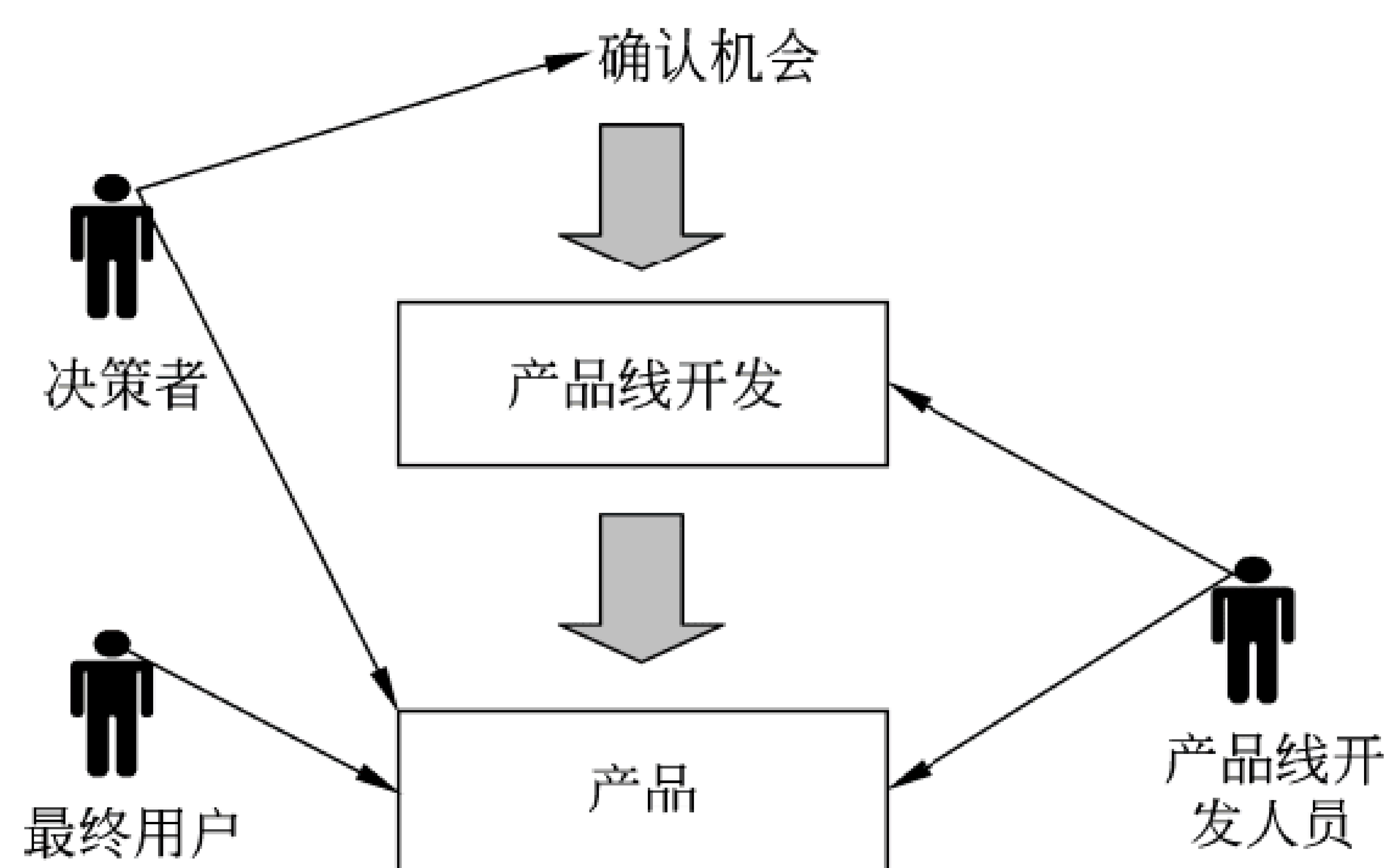


图17-6　关键干系人的观点

3．需求建模

在开始启动产品线分析时，需要回答以下几个基本问题：

（1）将要开发的产品线是否与组织的任务、商业目标和约束保持一致？

（2）产品线将由哪些产品组成？

（3）对组织来说，产品线的开发是否有意义？与之相关的成本、风险和利润是什么？

对这些问题的回答，取决于对目标市场特性的初步估计，期望的重用利益和诸如时间、经验和工具等资源的可用性。

产品线分析基于面向对象的分析、用例建模等。产品线需求模型是4个相互联系的工作产品的集合，如图17-7所示。

（1）用例模型（Use Case Model，UCM）：描述产品线干系人和他们与产品线的关键交互，干系人将验证产品线的可接受性。

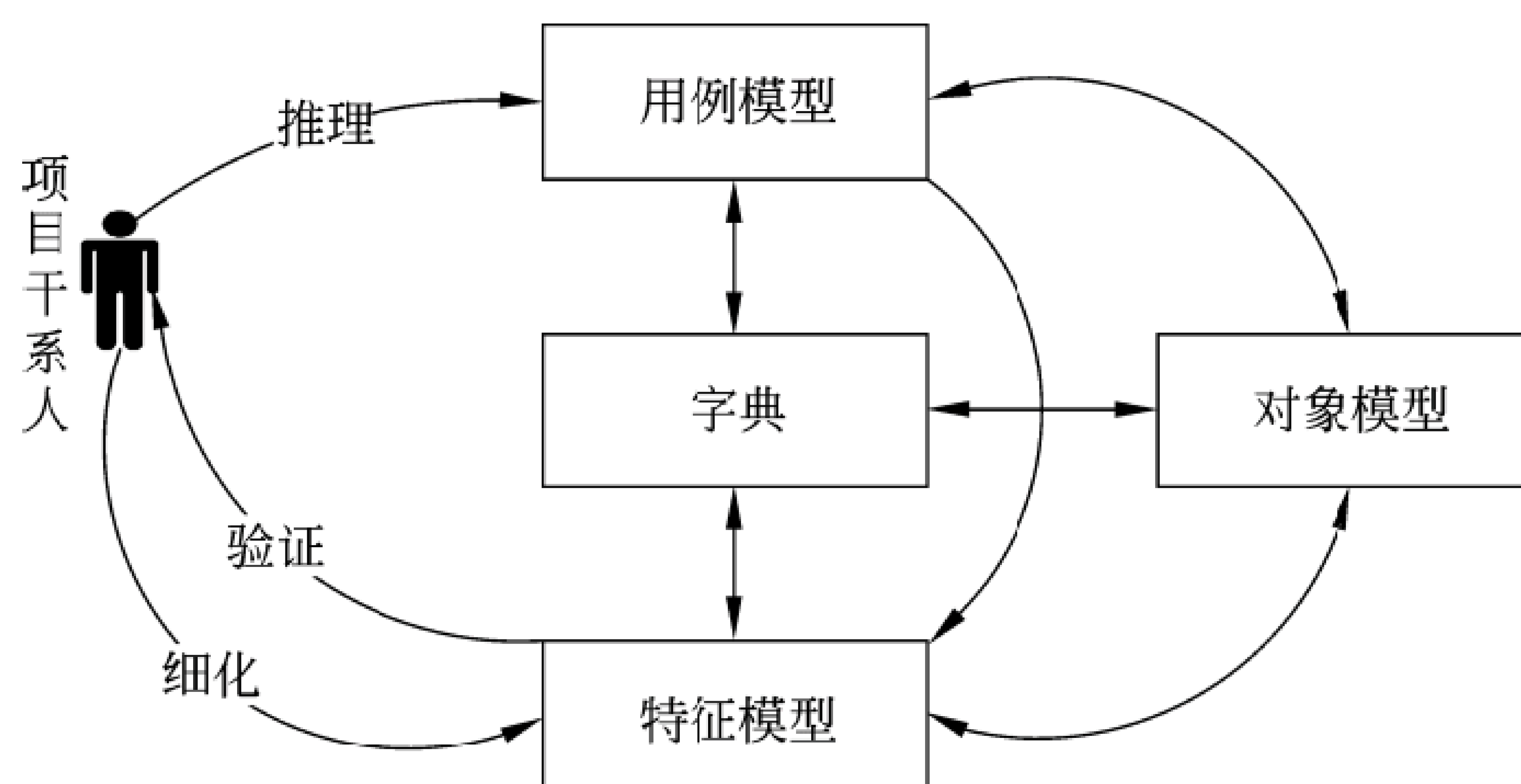


图 17-7　需求建模

（2）特征模型（Feature Model，FM）：描述产品线的干系人的观点。它捕获产品的功能特征和产品线及其产品的软件质量属性。

（3）对象模型（Object Model，OM）：描述产品线支持上述特征的功能，以及这些功能的通用性和可变性。

（4）字典（Dictionary）：定义用在工作产品中的，支持产品线需求的一致观点的术语。

需求模型支持发现最终用户和其他干系人的期望和需求，并把这些期望和需求文档化，提供影响产品线范围的早期和详细的信息，它是把干系人的需求映射为系列开发工作产品的基础，这种映射有利于决定和估计潜在的用户驱动（user-driven）的变更的影响。

17.6.3　产品开发

产品开发活动取决于产品线范围、核心资源库、产品计划和需求的输出，图 17-8 描述了它们之间的关系。

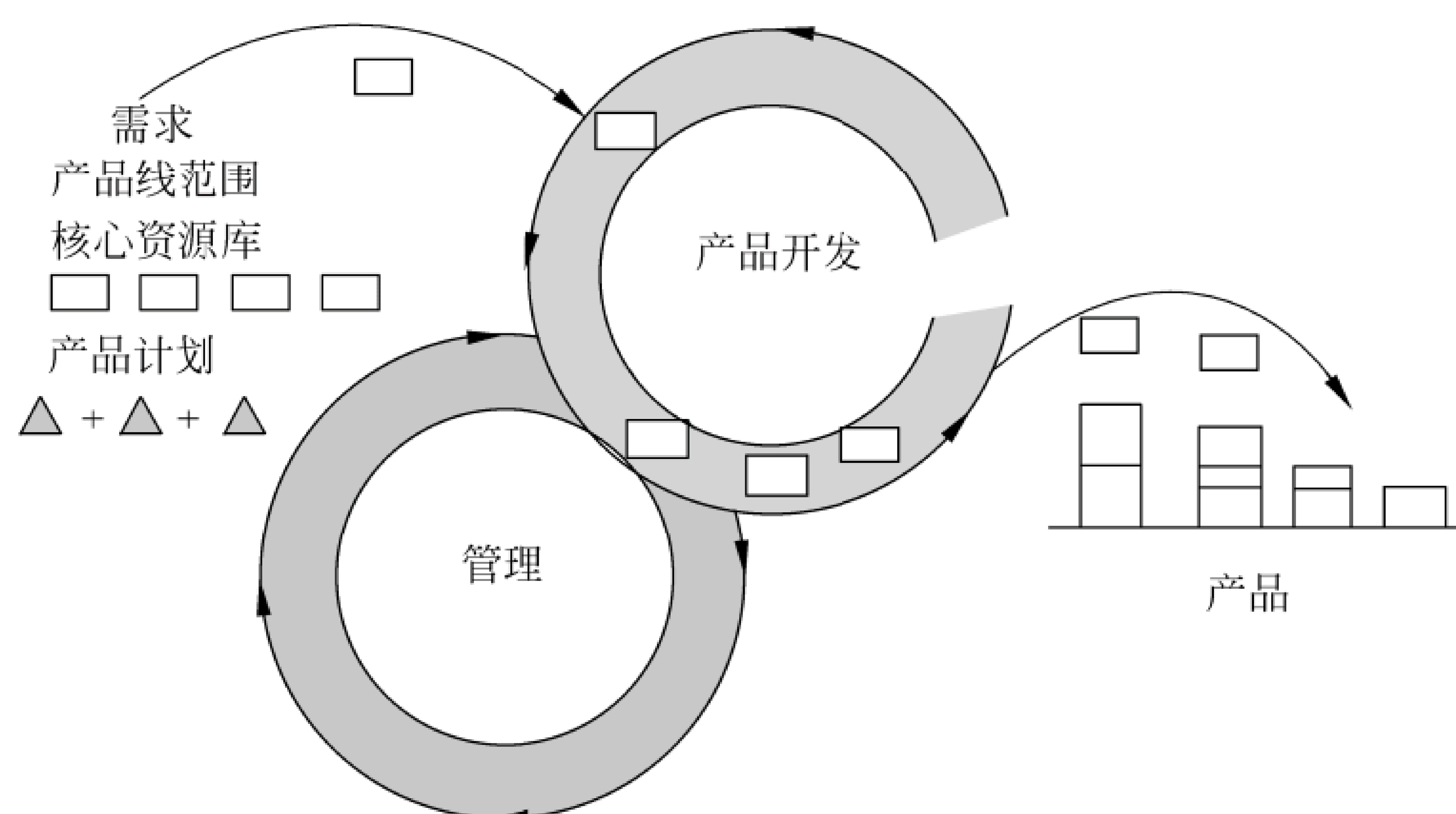


图 17-8　产品开发的输入与输出

产品开发的输入如下：

（1）特定产品的需求，通常由包含在产品线范围内的一些产品描述来表达。

（2）产品线范围，描述正在考虑的产品是否适合包含在产品线中。

（3）构建产品所需的核心资源库。

（4）产品计划，指明核心资源如何应用到产品的构建中。

希赛教育专家提示：产品线是一组相关产品的集合，但是，怎么实现却有很大的不同，这取决于资源、产品计划和组织环境。

17.7 软件产品线架构的设计

软件架构设计的主要目的是满足对软件的质量需求。本书的第 11 章详细讨论了软件架构的方方面面，但其讨论的范围是针对一个独立的软件系统而言的，本节主要讨论如何设计软件产品线的架构。

17.7.1 产品线架构概述

产品线的软件架构是产品线所有可重用的核心资源中最重要的部分，是软件产品线成功的关键技术性资源。产品线架构将用于产品线中所有产品，需要使每个产品的架构都能符合它的行为特性、性能和其他质量属性需求。

1．软件架构的应用方式

软件架构的应用方式主要有三种：用于独立软件系统、软件产品线架构、用于公共构件市场的标准软件架构。

独立软件系统的架构是常规软件开发周期的一部分，建立在独立软件系统的需求抽取和说明上，随后是详细设计、实现、测试等。

软件产品线架构指一个软件开发组织为一组相关应用或产品建立的公共架构。鉴于产品线软件开发在提高软件生产率和质量、缩短开发时间、降低总开发成本的重要作用，产品线架构又是软件产品线核心资源中最主要部分之一。面向独立软件系统的软件架构设计方法并不完全适用于软件产品线架构的设计，因为它们一般没有考虑产品线中不同产品之间的共性和个性问题。产品线架构可以使软件开发组织将总成本均摊到产品线的多个产品的设计开发中，从而充分地降低整体成本、有效地提高软件生产率。

标准软件架构主要是一个特定的领域中为构件开发者和构件使用者之间提供一个与构件的基础框架相关的“架构协议”，该协议主要描述了构件的功能、提供的和需要的接口、构件之间的依赖关系等。有时也将这些标准软件架构称为“构件框架”。标准软件架构可以分为两类：由某个标准化组织制定的“公共的标准软件架构”和由某个领域中占主导地位的组织或公司制定的“专有的标准软件架构”，或称为“工业标准软件架构”。OMG 制定的 OMA 以及为专门的应用领域（如医疗、电信等）制定的领域接口规范就是

一种公共的标准软件架构。

所有的软件架构都是抽象的，它们都允许有多个实例。独立软件系统的软件架构对架构的变化没有说明和限制。在架构实例化过程中，为了满足目标系统的行为和质量目标需求，几乎允许对架构进行任意的变化。软件产品线架构作为产品线中所有产品共享的架构和各个产品的架构的导出基础，必须在软件产品线架构中对允许进行的变化进行显式的说明和限定，最终的实例化结果才能既保持领域共性，又能满足特定产品的需求。

2．需要考虑的问题

与领域模型一样，软件产品线架构中也可以分为共性部分和个性部分。共性部分是产品线中所有产品在架构上共享的部分，是不可改变的。个性部分指产品线架构可以变化的部分。产品线架构的设计目的就是尽量扩展产品线中所有产品共享的共性部分，同时提供一个尽量灵活的架构变化机制。产品线架构主要需考虑以下因产品线的特殊性而出现的变化需求：

（1）产品线的产品有着不同的质量属性。例如，一个产品需要高度安全但运算速度要求低，另一产品可能需要运算速度快但对安全没有特别要求，产品线架构需要足够灵活来支持两个产品。

（2）产品之间的差异可能体现在各个方面：行为、质量属性、平台和中间件技术、网络、物理配置、规模等，产品线架构需要对这些差异进行处理。

有多种技术支持架构的变化。例如，采用构建时（build-time）对构件、子系统的参数组合进行设置来适应产品线变化，但该方法假设所有的变化都是可预测的，并且所有变化在构件的代码中都要实现，每一组参数组合对应一个产品的实现。

在面向对象系统中可以用继承和动态绑定等面向对象技术将类设计为在不同的产品中能对变化点进行不同说明实现。面向对象框架是这类技术的集中体现。也可以在变化点用构件替换来实现所希望的变化，这实际上是一种构件组合方式的变形。

3．个性实现机制

Mikael Svahnberg 和 Jan Bosch 对软件产品线架构个性的实现机制总结如下。

（1）继承：用于对象方法在产品中的不同实现和扩展。

（2）扩展和扩展点：通过增加行为和功能扩展构件的某些部分。

（3）参数化：用于构件的行为特征可以抽象并在构件构建时可确定的情况，例如，宏定义和模板都是参数化方法的一种。

（4）配置和模块互连语言：用于定义系统构造时结构和构件的选择方式和结果。

（5）自动生成：用更高级的语言来定义构件的特征，并自动生成相应的构件。

（6）编译时（compile-time）不同实现的选择：用于构件的变化可以通过选择不同代码段实现的情况，如 ifdef。

产品线架构是产品线核心资源的早期和主要部分，在产品线的生命周期中，产品线架构应该保持相对小和缓慢的变化以便在生命周期中尽量保持一致。产品线架构要明确

定义核心资源库中软件构件集合及其相关文档。

在各个产品的应用工程中，产品线架构被用来导出产品的架构。此时，如果发现有新变化点或产品线架构不能满足的需求模式，需要将这些信息反馈给产品线架构设计师，由他们决定是否对产品线架构进行修改并实施。

4．面临的困难

产品线架构设计面临的主要困难和问题如下。

（1）没有熟练的架构设计师：架构的设计还是一个不成熟的领域，设计更多地依赖于架构师的经验，而不是已经规范定义的规则、惯例和模式集合。尤其是在某个特定的应用领域中，该问题可能更加严重。

（2）参数化问题：参数化是一个支持产品线架构变化的有效的方法，但是，过于参数化易使系统难以使用和理解，参数化过少又会限制系统的变化能力。过早的参数绑定易使变化困难，绑定过晚（例如，运行时刻的动态绑定）易导致性能降低。

（3）必须有良好的领域分析和产品规则基础作保证，对技术发展趋势要做出准确预测，还要注意吸取相关领域的教训。

（4）软件开发、管理和市场人员组织的管理和文化对基于软件架构开发的适应程度。

（5）目前，支持软件架构设计的 CASE 工具较少。

（6）产品线架构设计师和产品开发者之间的沟通。

17.7.2 产品线架构的标准化和定制

产品线架构的设计有两种方式：使用标准架构和架构定制。作为标准，会有众多的软件开发组织遵循它，开发各自的应用或为该架构提供基础构件和应用开发的辅助工具等。采用标准架构建立产品线的软件架构，可以获得第三方软件开发组织的支持、有效地缩短开发时间、提高产品的可靠性和与同类系统的可集成性等。所以，如果产品线所在领域有相应的架构标准，应该尽量遵循它。

在宏观架构上，对标准的遵循比较容易。以层次架构为例，为适应应用的规模增大、复杂度提高，软件技术不断发展，相继出现了中间件技术、软件产品线等。应用的宏观架构自然地形成了“硬件→网络与操作系统→中间件平台→领域核心资源→应用”这样一个层次软件架构。

OMG 制定的软件架构标准 OMA 就是一个层次架构在面向对象环境中的演变，其层次为“ORB→公共服务对象→公共设施对象→领域对象→应用对象”。这个面向对象的层次结构和 OMA 的对象框架为产品线的宏观架构提供了很好的参考标准。另外，选用了 OMA 也应意味着选择了 CORBA 中间件平台，也同时获得了 OMG 相关标准和规范支持（如 UML、MOF、XMI、MDA 等）。

对于这个层次结构的低层部分（如公共服务、公共设施等）的标准遵循比较容易：中间件技术已经成为当前软件开发的主流技术，几个主流的中间件平台在相关的公共服

务标准上也出现融合的趋势。但该结构越是高层的部分，与特定领域和应用的相关性越大，在产品线架构设计中要遵循这些高层的标准就比较困难。因为 OMG 是以通用目的建立架构标准和领域接口规定的，面对的是整个软件应用领域，所以 OMG 划分的领域范围一般比较大。

另外，标准的制定只能针对应用领域当前的普遍情况，对快速变化的需求有一个逐步调整的过程。而某个软件开发组织的软件产品线的范围的确定要考虑市场需求，该组织的技术、文化、管理背景，所在领域的现状和发展趋势等多方面，很难做到和某个标准组织定义的应用领域范围一致。

架构风格是一个使产品和产品线具有良好的可移植性的结构，产品和产品线通过最小的修改就可移植到一个新的平台上。这里的平台包括硬件平台、操作系统、网络系统、中间件环境等。如果产品线中的产品需要运行在不同的平台上，或整个产品线也有可能移植到一个新平台上的话，层次架构则是产品线架构最好的选择。

综上所述，在产品线的宏观软件架构中，笔者建议使用层次架构。在该层次架构中，公共服务和设施及其以下层次遵循标准架构；在领域层以标准架构为参考，在应用层根据应用的特定需求进行定制。

17.8　软件产品线架构的演化

产品线架构就是一个软件架构和一组在一族产品中可重用的构件，为增加软件重用、为降低软件开发和维护的成本提供了一条重要的途径。

产品线架构中的软件一旦开发出来，就要经历演化，因为新的需求总是在不断地出现。因为需要处理如此多的需求（这些需求甚至可能是自相矛盾的），通常的处理方法是创建两个独立的演化周期。也就是说，对每个产品而言，需要合并产品特定的需求；对整个产品线架构而言，需要合并影响整个产品线中所有产品或大多数产品的需求。

产品线的演化是由需求变更驱动的，这些需求的变更可以来自多个方面，如市场、企业将来的需要或在产品线中引入新的产品等。产品线的演化可分为两个部分，一是企业如何组织其产品线结构的变化；另一个是实际的演化，该演化作为一个需求通过静态组织进行传播。

考虑图 17-9 所示的组织结构图，其中特殊的业务部门由数个需求驱动。这些需求在该业务部门负责的产品和整个产品线的一般需求之间进行划分。产品线架构即可能影响一个特殊的架构框架，从而在其接口上创建一个变更，也可能引起该框架的一个或多个具体实现的改变。在某些情况下，需求甚至可以使一个构件分解为两个构件，或在产品线架构中引入一个全新的构件，产品线架构和数个具体的框架实现实例化为一组产品。

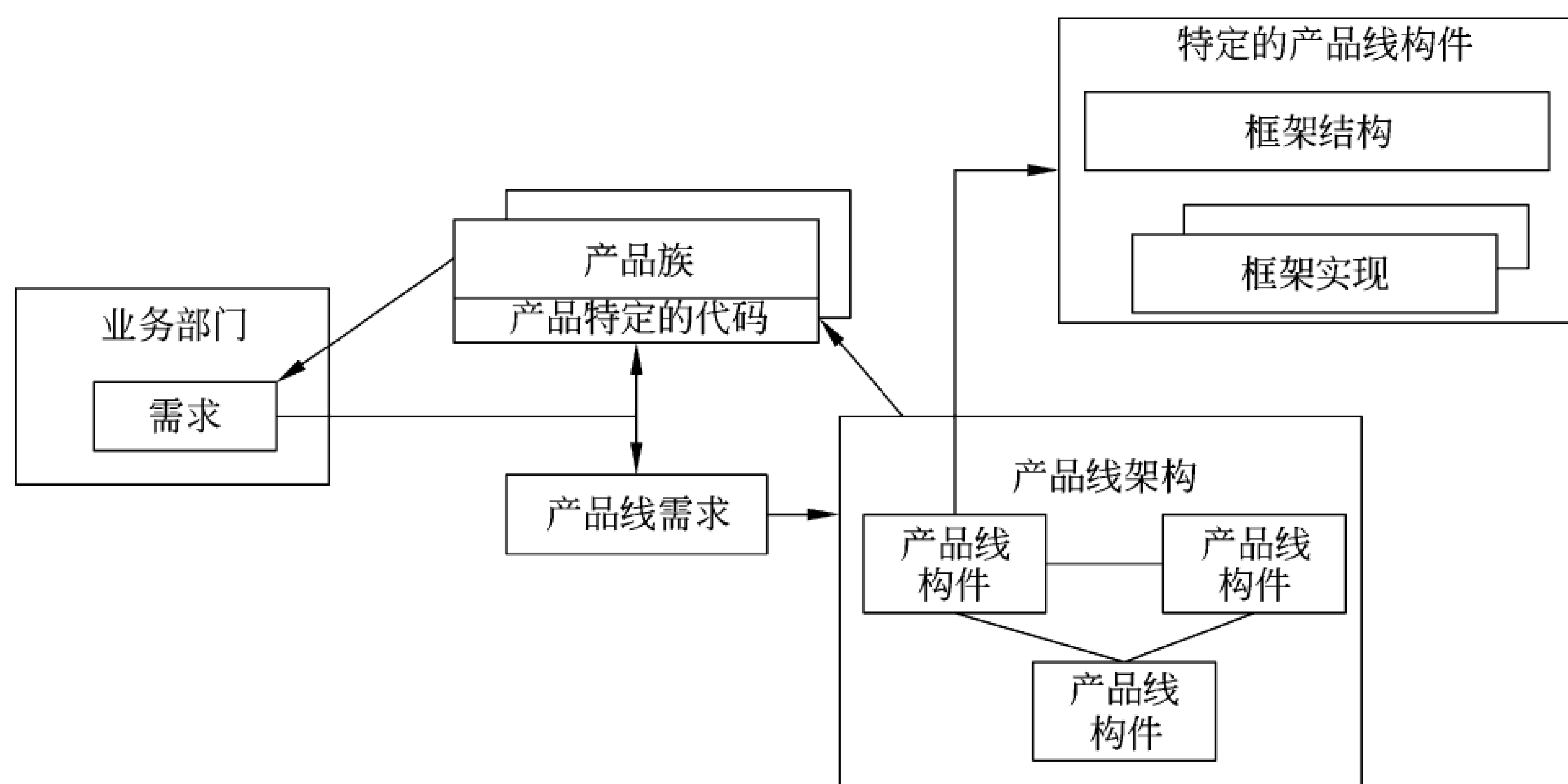


图 17-9　产品线的演化

下面，通过一个案例讨论产品线架构的演化过程。

17.8.1　背景介绍

本节选择瑞典的 Axis 通信公司的产品线架构作为讨论案例。Axis 公司是一家相对较大的软件和硬件企业，专业从事网络设备的开发。从单一的产品（IBM 打印服务器）开始，现在，已经延伸到包括摄像服务器、扫描服务器、光盘服务器以及其他的存储服务器在内的产品线。Axis 公司在 20 世纪 90 年代就开始使用产品线方法，他们的软件产品线由一组大小不一的、可重用的、面向对象的框架组成。他们的产品线架构是作为不同的产品、产品类型及产品族的等级结构形式出现的，如图 17-10 所示。每个产品族都由一个业务部门进行维护，维护和演化产品线架构，使之满足特定部门的需要。

按照 Axis 公司的观点，产品线架构是由构件及其关系组成的。构件在自己的面向对象的框架中，框架管理一些特殊的函数。如上所述，Axis 的框架由一个抽象的架构和数个具体实现组成，框架的实例通过继承抽象类的具体实现来创建。

本节的目的是讨论框架作为产品线的一部分的演化过程，为此，不失一般性，把讨论焦点放在存储服务器架构上。存储服务器是一个网络光盘设备产品，后来把 Jaz 服务器和磁盘服务器也包括了进来，这些产品的核心是一个文件系统框架，该框架允许统一存取所有类型的存储设备。

Axis 公司的文件系统框架已经有两代明显区别的产品，第一代产品是在光盘服务器中，所以设计为只读类型的文件系统，第二代产品从一开始就设计为可读写的。第一代产品包含一个框架接口，通过该接口可以创建不同文件系统的具体实现，这些具体实现还提供了块设备接口和小型计算机系统接口（Small Computer Standard Interface，SCSI）

接口。与这个抽象框架并行的是一个存取控制框架，在框架接口的顶部，增加了不同的网络协议。例如，网络文件系统（Network File System，NFS）、服务器信息块（Server Message Block，SMB）和 Novell Netware。如图 17-11 所示。

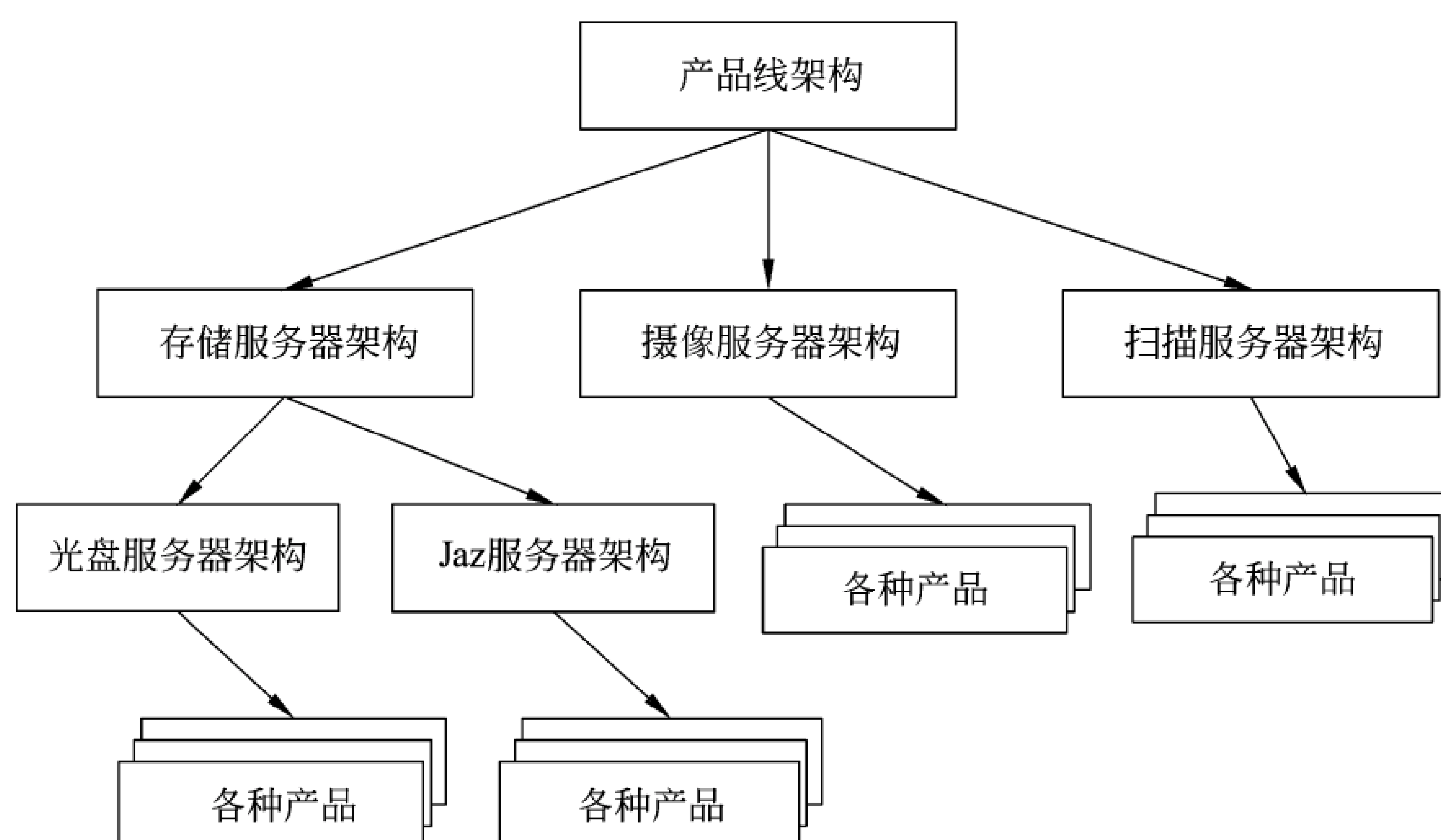


图 17-10　产品线架构的等级

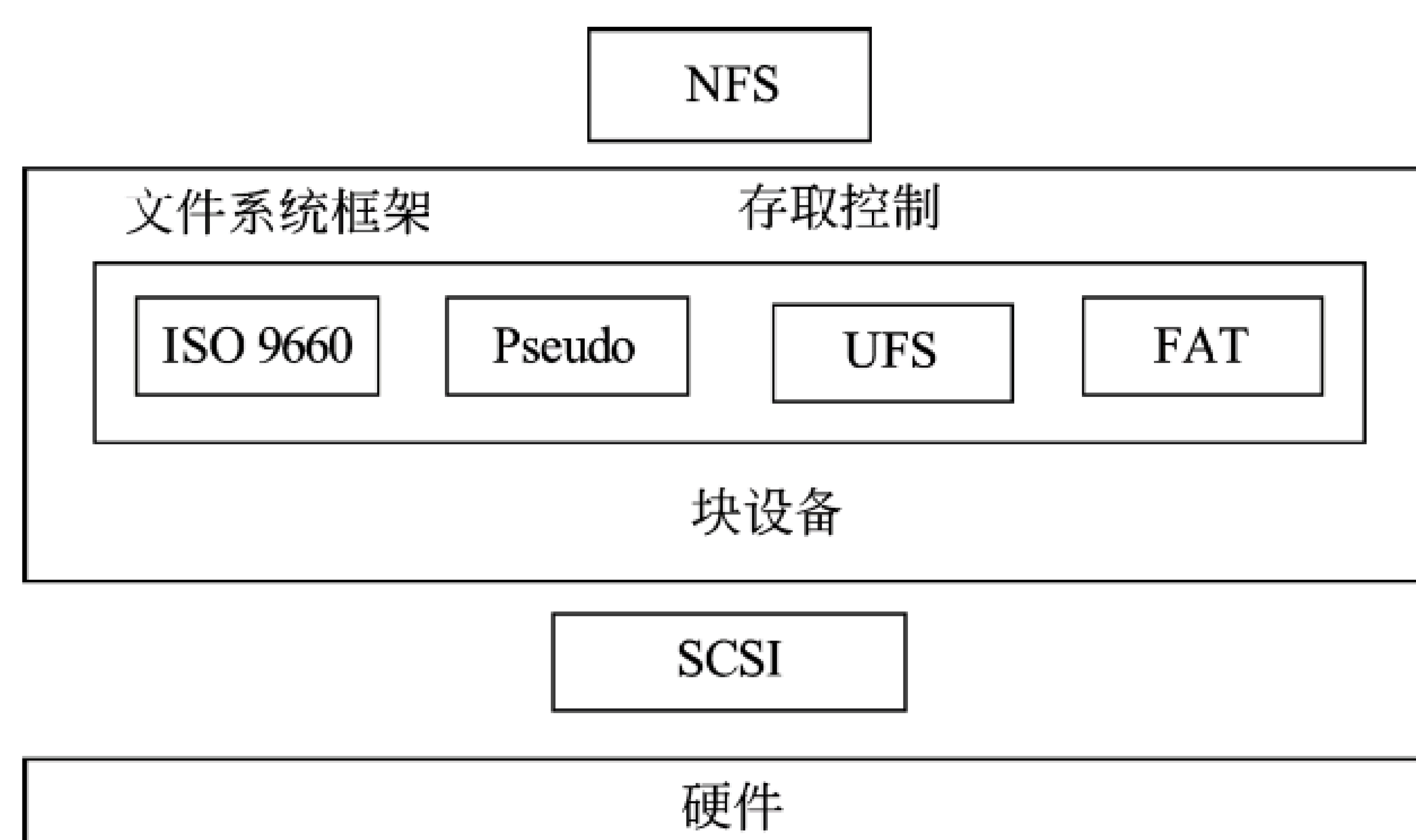


图 17-11　第一代文件系统框架

第二代产品在很大程度上与第一代产品类似，但做得更加模块化，从一开始就预见了系统将来可能的功能增强，因为这已经在第一代产品中发生过。第二代文件系统框架如图 17-12 所示。可以看出，该框架可以划分为更小的和更专业化的框架。值得注意的是，存取控制部分也分离成一个单独的框架。

17.8.2　各种产品版本

本节将介绍两代产品的各种主要发行版本。

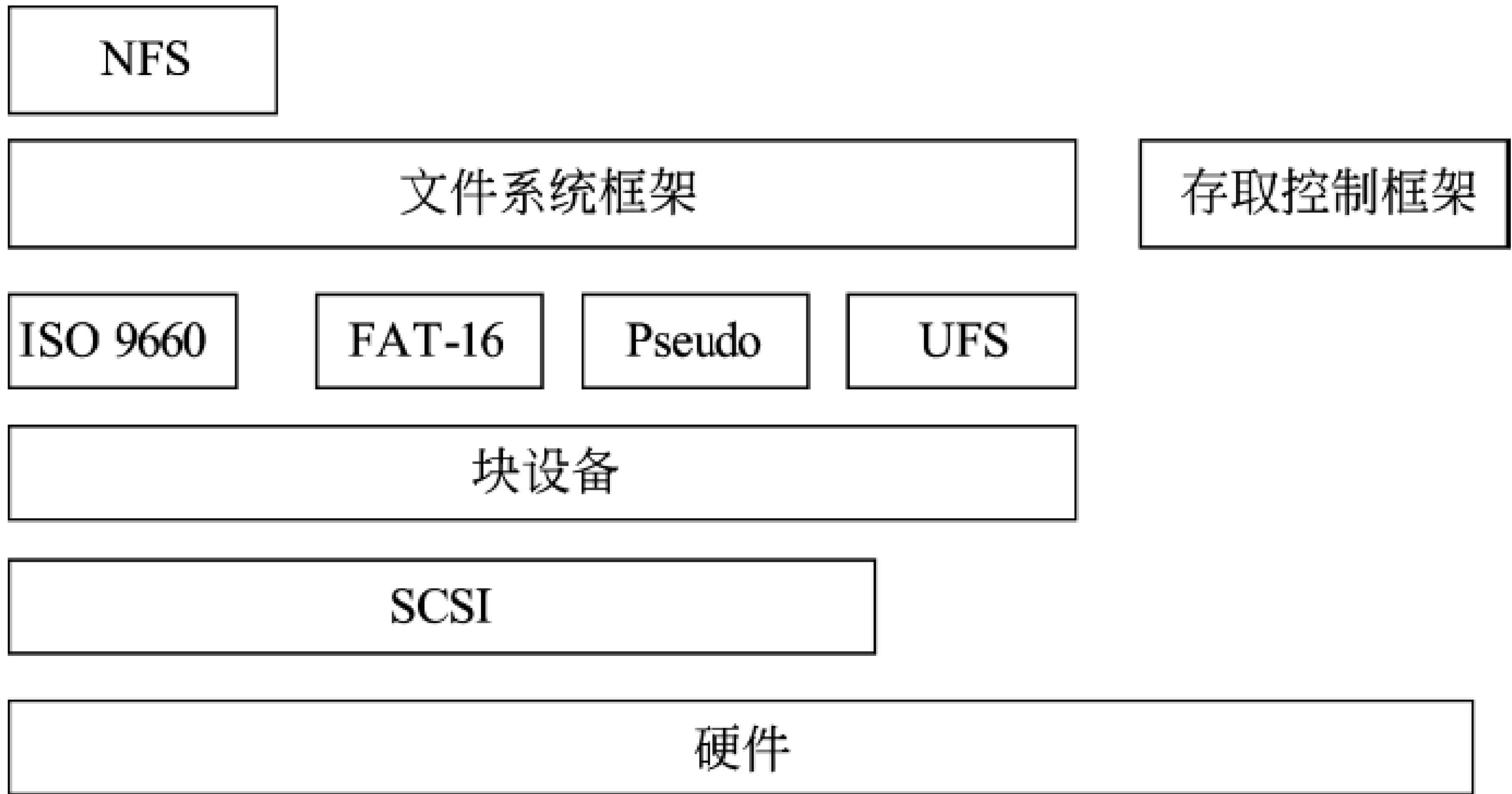


图 17-12　第二代文件系统框架

1．第一代产品

第一代产品有 4 个主要发行版本。

（1）版本 1。在第一个版本中，主要用来支持光盘服务器，支持网络通信、网络文件系统、光盘文件系统，能够存取光盘硬件。文件系统支持 ISO 9660 文件系统，同时为了控制和配置的目的，系统还支持虚拟 pseudo 文件系统。支持的网络文件系统有 NFS 和 SMB。在图 17-11 中，NFS 作为网络文件系统的示例为文件系统框架提供了接口，文件系统框架又为硬件存取提供了接口（SCSI 接口）。

（2）版本 2。发行第二个版本的目的是创建一个新的产品，使之支持令牌网（Token Ring，TR）以代替第一个版本中的以太网（Ethernet）。增加了对 Netware 文件系统的支持，对 SMB 协议进行了扩展，而且设计了 SCSI 模块。另外，在第二个版本中引入了名字空间缓冲（Namespace Cache，NC）。图 17-13 描述了第二个版本的改变情况。

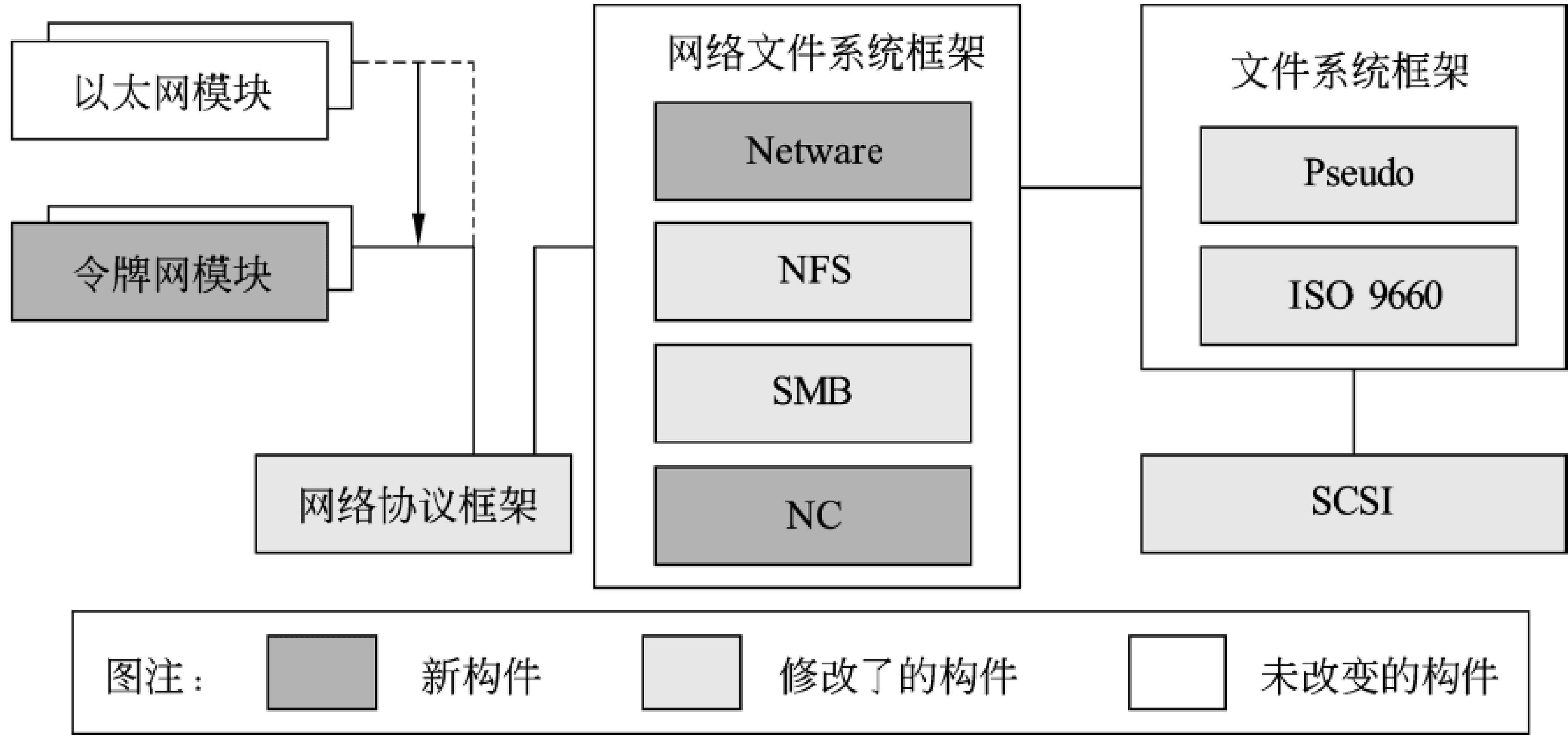


图 17-13　第一代产品的第二个版本

（3）版本 3。发行产品线架构的第三个版本的目的是整理和修改以前版本中存在的问题。修改了 SCSI 驱动以支持新版本的硬件，同时，增加了一个 Web 接口用来浏览 pseudo

文件系统，在 ISO 9660 模块中增加了对长文件名的支持。图 17-14 描述了第三个版本的改变情况。

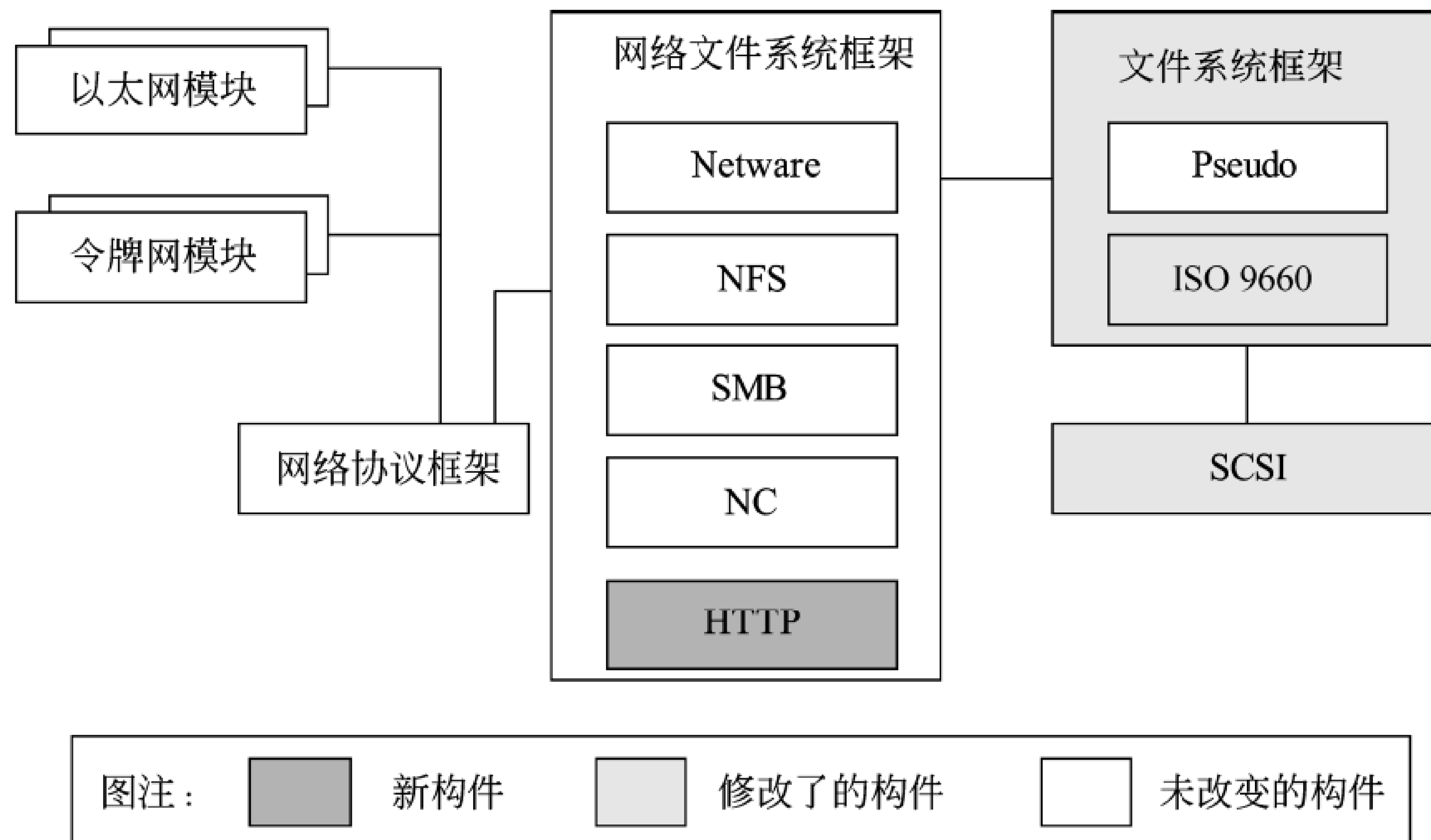


图 17-14　第一代产品的第三个版本

（4）版本 4。第四个版本是第一代产品的最后一个版本，在这个版本中，增加了对 Novell 目录服务（Novell Directory Service，NDS）的支持，同时改进了对 Netware 协议的支持。NDS 需要新的算法来获取存取文件的权限，因此，必须修改其他所有的网络文件系统的接口，使之为这种新算法提供支持。同时，去掉了在第二个版本中引入的名字空间缓冲。图 17-15 描述了第四个版本的改变情况。

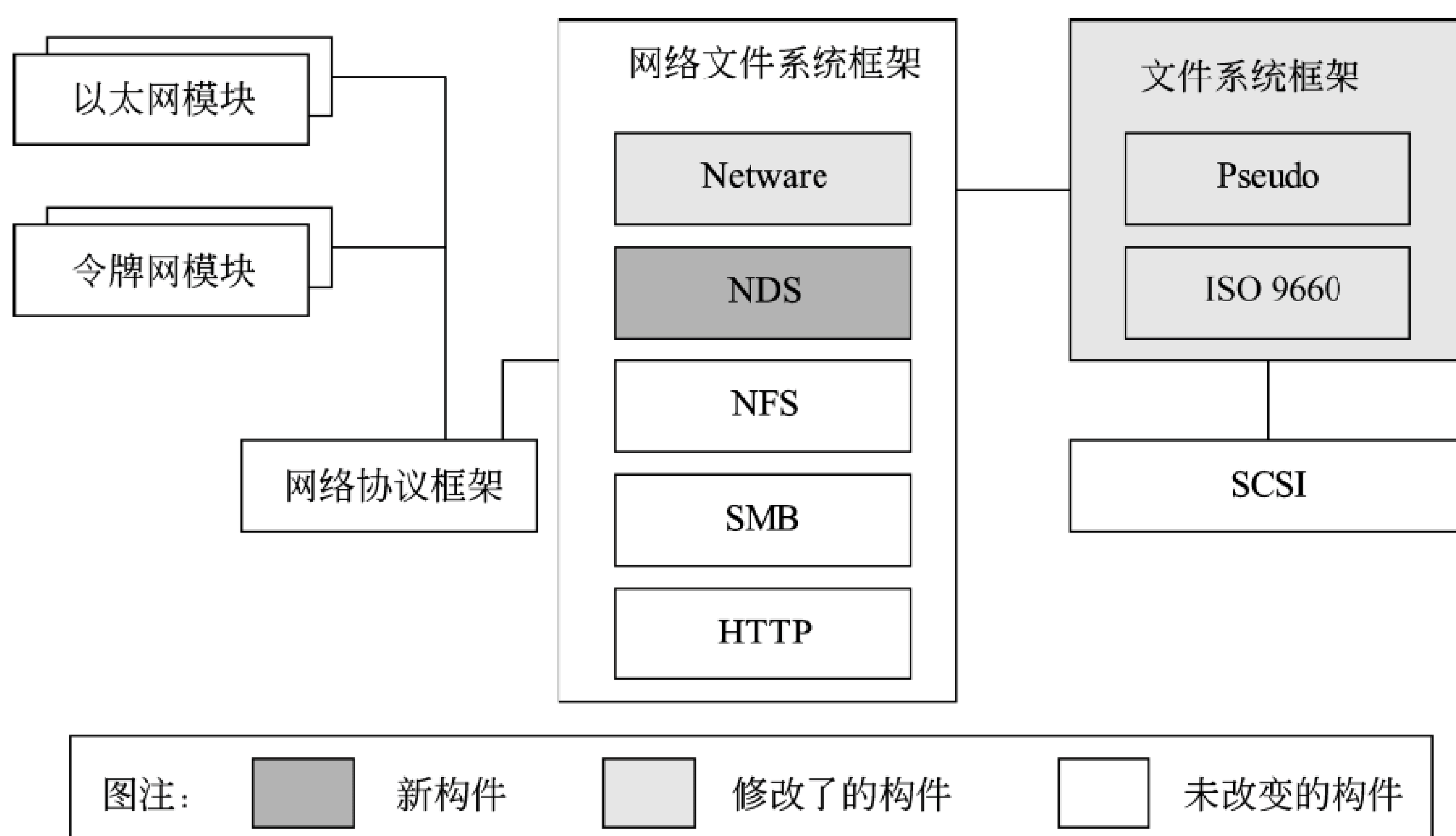


图 17-15　第一代产品的第四个版本

2．第二代产品

第二代产品与第一代产品几乎是同时开始开发的，如图 17-16 所示。两代产品同时并存了几乎 4 年的时间，但是当第一代产品的第四个版本发布后，所有开发人员和其他资源都转向了第二代产品，所以，实际并行开发的时间只有两年多一点。图 17-16 中的箭头表示资源的转移。

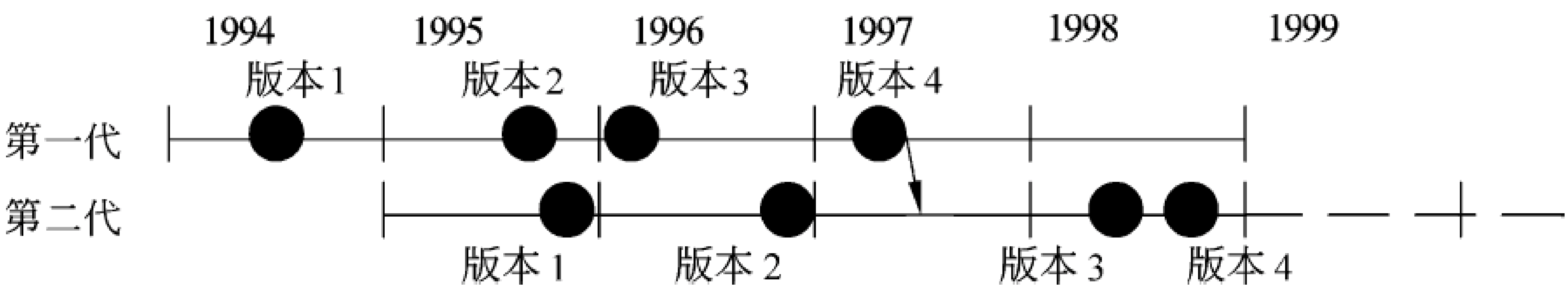


图 17-16　文件系统框架的时间线

（1）版本 1。第二代产品的第一个版本的需求与第一代产品的第一个版本十分类似，所不同的是第二代产品从一开始就是要开发可读写的系统。这时，利用了第一代产品开发中的经验。从图 17-11 和图 17-12 中，我们可以看出，第二代产品从一开始就注重了模块化。在第一个版本中，只支持 NFS 和 SMB，与第一代产品相比，这里增加了写的功能。另外，为了理解基于节点的文件系统，还开发了一个私有文件系统 MUPP。

（2）版本 2。在第二个版本中，增加了对 FAT-16 文件系统的支持，删除了第一个版本中的 MUPP 文件系统、NFS 协议，这样，系统只支持 SMB 协议。另外，对 SCSI 模块和块设备模块也作了一些修改，如图 17-17 所示。因为在第二代产品中，架构中的可变部分被分离成几个框架，所以实际的文件系统框架架构仍然保持不变。

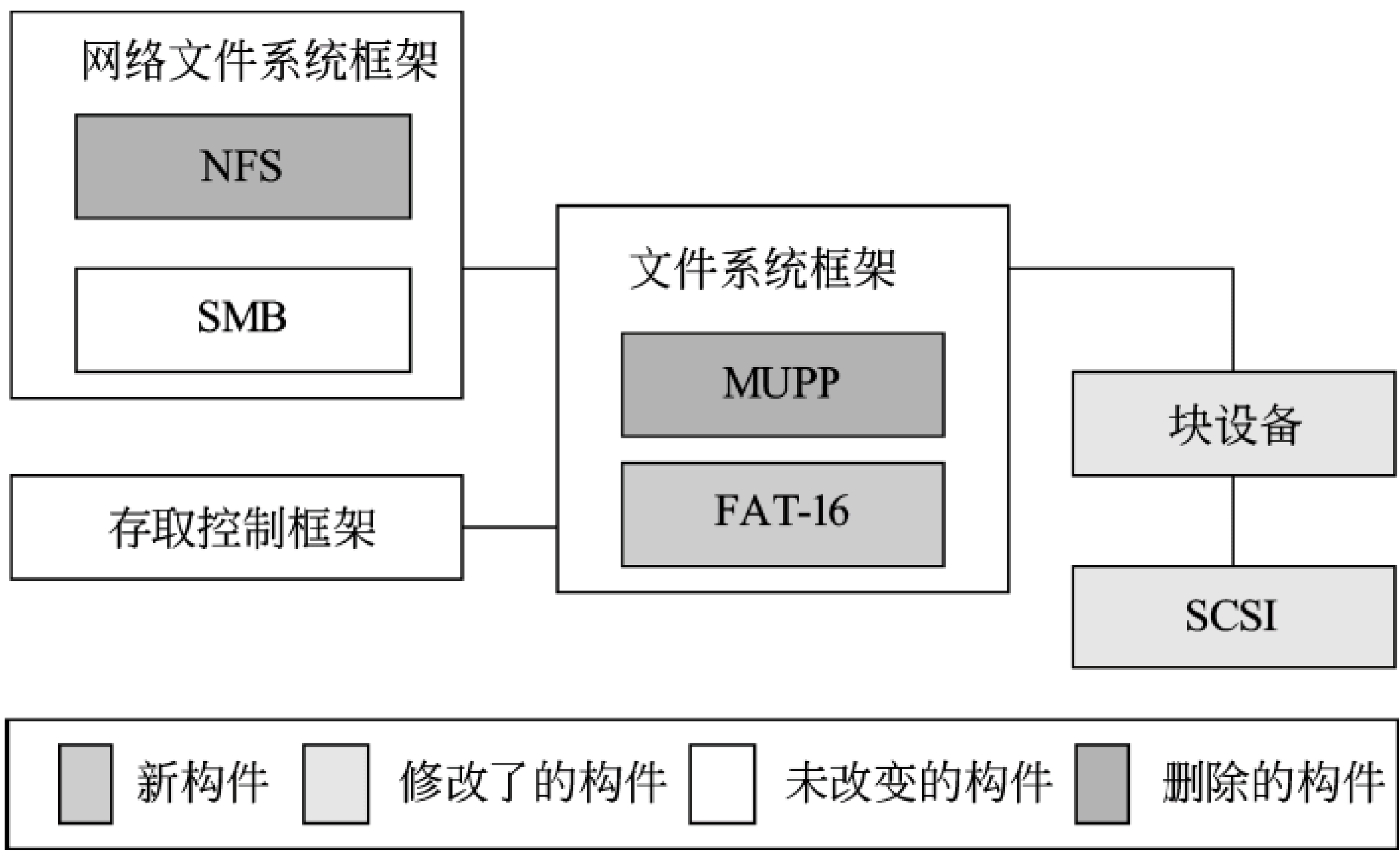


图 17-17　第二代产品的第二个版本

（3）版本 3。第三个版本的需求来自于开发一个支持备份和冗余磁盘阵列（Redundant Array of Independent Disks，RAID）的硬盘服务器。为了支持磁带备份，增加了一个新的文件系统模量传输函数（Modulation Transfer Function，MTF），并且决定增加对基于

节点的文件系统统一光盘格式（Universal Disc Format，UDF）的支持，SMB 和 Netware 用来支持网络文件系统。另外，新版本还支持网络管理协议 SNMP。MTF 是作为一个新的文件系统来开发的，而 Netware 是从第一代产品中复制过来的。为了与新的网络文件系统、Netware、HTTP 和简单网络管理协议（Simple Network Management Protocol，SNMP）等协同工作，也修改了存取控制框架。最后，块设备被改名为存储接口，如图 17-18 所示。

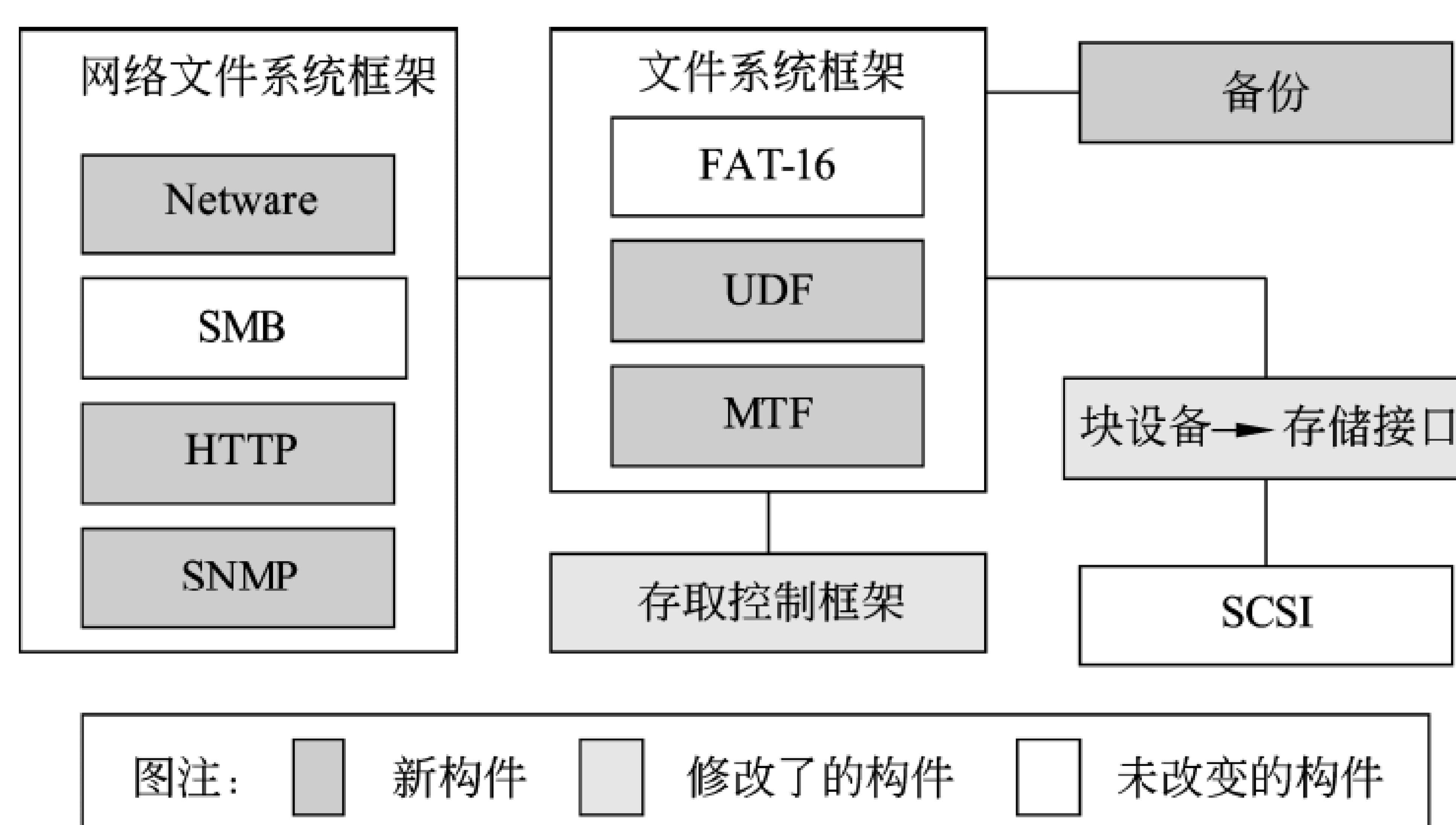


图 17-18　第二代产品的第三个版本

（4）版本 4。第四个版本主要是为了开发一个与光盘服务器协同工作的光盘转换器，为了满足这个需求，做了一些小的改动。在文件系统构件中，实现了使 ISO 9660 支持两种方式（Rockridge 和 Joliet）的长文件名，在网络文件系统中，又重新引入了 NFS 协议。抛弃了原来的存取控制框架，重新编写了新的存取控制框架。如图 17-19 所示。

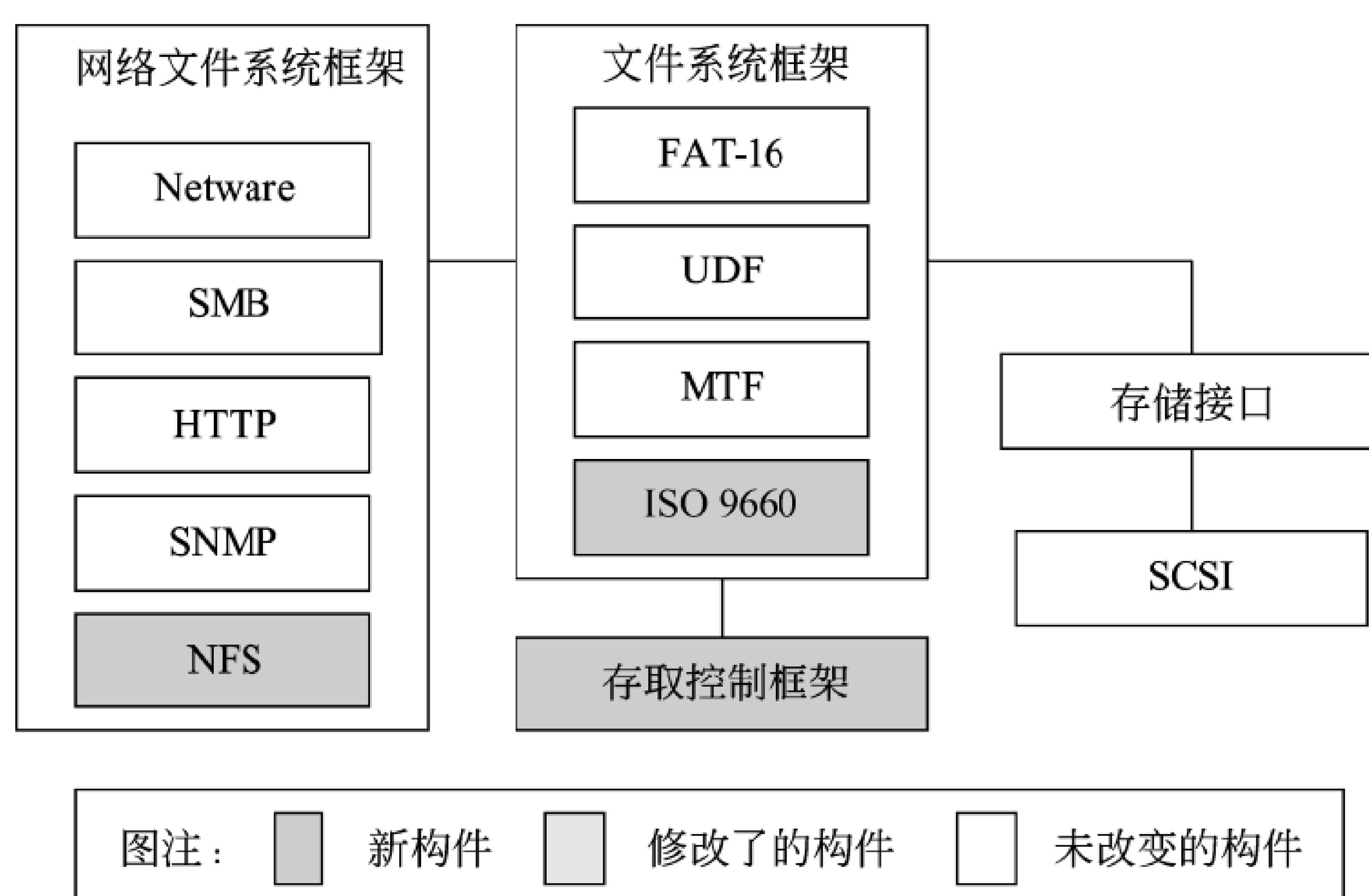


图 17-19　第二代产品的第四个版本

在很多情况下，演化类型并不是由需求变化直接引起的，而是需求变化的间接结果，一种类型的变化引起另一种类型的变化，后者与前者往往处于同一个级别，即在产品线架构中的变化会引起另一个产品线架构类型的变化。但也有例外，例如，一个新的框架实现可以导致另一个框架实现的改变，从而改变两个相关构件之间的关系。

本章参考文献

[1] 张友生. 软件体系结构第 2 版. 北京：清华大学出版社，2006

[2] 王广昌. 软件产品线关键方法与技术研究. 浙江大学博士学位论文，2001

[3] J. Kuusela, Juha Savolainen. Requirements engineering for product families. Proceedings of the International Conference on Software Engineering. Limerick, Ireland, 2000

[4] Greg Butler. Object-oriented application frameworks. http://www.cs.concordia.ca/~faculty/gregb

[5] Mohamed Fayad and Douglas C. Schmidt. Object-oriented application frameworks. Communications of the ACM, 1997,40（10）：32-38

[6] Martin Griss. Domain engineering and reuse. IEEE Computer, Roundtable on Software Development Trends. 1999

[7] Martin Griss. Implementing product-line features with component reuse. Proceedings of 6th International Conference on Software Reuse, SpringerVerlag, Vienna, Austria, 2000

[8] Weiss David M and Chi Tau Lai. Software product-line engineering: a family-based software development approach. Addison-Wesley, 1999

[9] Jan Bosch. Software product lines: organizational alternatives, the 23rd International Conference on Software Engineering (ICSE 2001), 2000

[10] D. Batory and Y. Smaragdakis. Object-oriented frameworks and product-lines. 1st Software Product-Line Conference, Denver, Colorado, 1999

[11] M. Svahnberg and J. Bosch. A case study on product line architecture evolution. http://www.ide.hk-r.se/~msv/

[12] J. Bosch. Evolution and composition of reusable assets in product-line architectures: a case study. The 1st Working IFIP Conference on Software Architecture, 1998

第 18 章　敏 捷 方 法

传统的软件工程方法，无论是经典的软件开发模型，还是统一过程和快速应用开发（Rapid Application Development，RAD），都强调软件文档的重要性，而敏捷方法论有一个共同的特点，那就是都将矛头指向了文档，它们认为传统的软件工程方法文档量太“重”了，谓之为“重量级”方法，而相应的敏捷方法则是“轻量级”方法。

18.1　敏捷宣言

2001 年初，一批志同道合的业界专家聚集在一起，将各自在轻量级方法论研究的基础上共同交流，以期解救那些陷入了不断增长的软件过程泥潭中的软件开发团队。也正是在这次大会上，找到了一个新名词 Agile 来描述他们的方法论，并且成立了敏捷联盟。该联盟的核心成员一同创建出了一份具有巨大影响力的价值声明，也就是敏捷联盟宣言，该宣言主要包括以下 4 个方面。

1．个体和交流胜过过程和工具

敏捷联盟成员一致认为，在软件开发行业的人、过程、工具三个环节中，最重要的一环是人。如果没有优秀的成员，那么再好的工具、再好的过程也是无法保证项目成功的。当然，这也不是说过程与工具不重要。工具可以很有效地武装开发人员，以提高他们的产能，使其更加优秀；而定义良好的过程能够保证项目更加有序，不好的过程将会对团队带来很大的副作用，使优秀的成员也无法充分发挥自己的作用。

要想充分发挥人的作用，交流是十分重要的，只有充分的交流、沟通，才能够将团队凝聚在一起，形成高效的团队。

2．可以工作的软件胜过面面俱到的文档

在以往的系统开发中，开发人员和维护人员被缺少文档的软件系统折腾得苦不堪言，缺乏有效和必要的文档使得软件变得更加难以维护。因此，各种文档规范越来越多、越来越细，逐渐将开发团队淹没在书山文海之中。

但是，对于客户而言，真正能够产生价值的东西是可以工作的软件，而非这些面面俱到的文档。文档的作用是帮助团队更高效、高质地交付软件系统，而绝非软件开发的目标。也就是说，即使文档再丰富，没有可以工作的软件，客户永远无法满意。因此，在软件开发过程中尽快地提供可以工作的软件，逐步迭代的交付中间成果，将是保证软件开发质量与效果的好办法。

3．用户合作胜过合同谈判

用户对软件系统的需求是各不相同的，即使是同一个领域，不同的用户也会有不同的要求。因此，软件开发的个性化、定制性很强，让用户参与到项目中，通过紧密的合作来实现项目开发，要比合同谈判效果好得多。合同是静态的、是不易变化的，而项目开发过程中需要用户及时的、频繁的反馈，因此，经常邀请用户到开发现场，就是成功的开始。

希赛教育专家提示：用户通常不愿意参与开发，认为这是开发组织的事。其实，要说服用户并不难，毕竟软件项目失败，最大的受害者不是软件开发组织，而是用户。

4．响应变化胜过遵循计划

“世界上唯一不变的就是变化”，没有一份计划能够与事实完全吻合，这种情况在软件项目中更是表现得淋漓尽致。面对这种现状，最好的办法不是违心的一味遵循计划，而应该对团队进行调整，做好响应变化的准备。

具体来说，就是在制订计划的时候采用逐步求精的办法，先将整个开发过程大致地分成几个阶段，并制订粗略的计划。然后以 1～2 周为周期对计划进行细化，这种迭代的计划制定过程，能够充分考虑到日新月异的变化。

18.2 敏捷原则

注重个体与交流，重点关注可以工作的软件，提高用户参与度，以积极的心态响应变化是敏捷方法论的核心价值观。为了贯彻这四大价值观，敏捷联盟提出了 12 条区别于重量级过程的原则。

（1）尽早，持续交付有价值的中间软件使用户满意。很多开发组织经常会在时间期限上进行没有原则地退让，其结果却是让用户一等再等，不仅没有按承诺兑现，甚至是时间超过一倍，但仍然不见软件的踪迹。这种不守信的状态，使得整个软件业走入了一个负螺旋发展。敏捷方法论提出了一种新的逻辑，将尽早、持续地交付可运行的中间成果，有价值的中间结果，使得客户能够尽早地、持续地了解到软件开发的进展，并且将需求的变化、系统的改进意见尽早地提出来，这会使得用户的满意度大大提高。

（2）即使到了开发后期，也欢迎需求变化，利用响应变化创造竞争优势。敏捷方法论鼓励团队拥抱变化，通过应用各种技术来提高软件结构的灵活性，本着简单的原则进行设计，以响应变化的能力作为团队的核心竞争力。

（3）经常交付可工作的软件，间隔时间可以是几周到几个月，间隔越短越好。由于敏捷方法论奉行用户合作、用户参与，而要让用户更加有效地参与，经常性、频繁地交付可工作的中间软件，将可以有效地加强开发人员与用户之间的沟通，从而将隐藏的需求变更及早触动。

（4）在整个开发过程中，业务人员和开发人员必须天天都在一起工作。在开发中，

不仅要经常邀请用户参与开发，还应该包括代表用户的业务人员。因此在开发人员、用户、业务人员等相关项目干系人之间建立频繁而且密切的交流与沟通，将是使项目保持高度灵活性的关键。

（5）为开发人员提供环境和支持，给予信任，以人为本地构建项目。敏捷方法论是崇尚“以人为本”精神的，认为项目成功的最关键因素是人，其意义超过过程和工具。建立一支优秀的团队，并在环境与精神上提供支持，给予信任，将是项目成功的关键。这也是与传统的以“过程”为主的管理思想的最大不同。

（6）团队内部，最有效的沟通方式莫过于面对面的交流。在重量级方法论中，人们尝试着通过编写规范、精美的文档进行交流；而在敏捷方法论中则更加重视的是开发团队成员之间的面对面交流，大家坐在一起，用一块白板，或是一张纸，一边绘制草图，一边交谈，这是最有效的沟通方式。

（7）工作的软件是度量进度的最首要标准。要衡量工作进度，采用的基点不是文档的完成情况、不是已完成的代码行数，而是可以工作的软件完成了多少功能、实现了多少用例。这是敏捷方法论的共同点，因为只有可工作的软件才是有价值的。

（8）提倡可持续的开发速度，责任人、开发者和用户应保持一个长期的、恒定的开发速度。软件开发绝不是短跑，它更像一场挑战耐力的马拉松长跑。因此，过早的冲刺、在前期过度的工作，将不利于项目按照持续的开发速度进行下去。因此，敏捷方法论反对加班，因为这样的行为会使得团队的精力过早耗尽，过早地对项目失去兴趣和信心，从而得到事与愿违的结果。

（9）不断关注好的技能和设计会增加敏捷能力。保持软件高质量、简洁、健壮，是实现快速软件开发的重要途径。因此，只有大家都致力于编写高质量的代码，不创造混乱，才能够提升敏捷能力。

（10）本质是简单，这是使未完成的工作最大化的艺术。不管明天的需求，只采用符合今天需求的简单设计。因为谁也不知道明天是怎么样的。变化太快了，今天的设计考虑太多明天的需求，就有可能做了过多的无用功。

（11）自组织的团队才能够做出最好的架构设计和需求分析。最优秀的团队不是被强权管理下的团队，而是形成了一个良好的协作，能够内部进行任务分解、协调的团队。

（12）团队应定期在如何更有效工作方面进行反省，然后对自己的行为做出改进。不断地回顾、总结，并从中找到团队未能最有效工作的瓶颈点和问题点，并且通过细致的分析与讨论，找到其要点，并做出相应的改进是十分重要的。

18.3　敏捷方法论

20 世纪 90 年代，随着互联网应用的普及，软件系统的变化周期更短了，因此，对于其灵活性的要求更高了，这就催生了大量的敏捷方法的诞生，包括极限编程方法、水

晶方法、动态系统开发方法、特征驱动的软件开发方法、自适应软件开发方法、Scrum方法。在这些敏捷方法中，最具代表性的是极限编程方法。

本节介绍后五种方法，有关极限编程方法，将在18.4节专门介绍。

18.3.1 水晶方法

Alistair Cockburn认为软件开发是一种发明与交流的合作性工作，他强调人、交互、团队、技能、才智和交流，并将其作为性能的第一要义。他认为过程虽然重要，但还是在于其次。正是由于其本着人和团队是第一要义，过程与工具是第二要义，因此，他认为每个团队都应该利用为他们量身定制的过程，而且过程应该被最小化，即够用就好。在这种思想的基础上，Alistair Cockburn创建了水晶（crystal）方法。

1．水晶方法论的背后

Alistair Cockburn为敏捷联盟做出的贡献不仅仅是水晶方法论本身，而是其关于方法论设计原理的总结。他认为："如果团队能够更快生产出可运行代码，或是能够利用成员间丰富的交流渠道，那么它们可以减少中间工作产品。另外，每个项目都很相似，但却进度拖期，那么该方法学，也就是该团队适应的那套惯例，必须被改革并发展"。

在设计水晶方法时，Alistair Cockburn引入了两个绝对的准则：使用的增量式循环不超过4个月，应用反思工作室促使方法学的自适应，从中敏捷的思想跃然于表。其将方法学归结到角色、技能、团队、技术、活动、过程、里程碑、工作产品、标准、工具、个性、质量、团队价值等13个基础元素中。正当大多数方法学把90%的时间花在与人无关的问题上的时候，Cockburn却鼓励改变重点，建议花费30%～70%的时间用于人的问题上。

2．水晶方法的框架

水晶方法的另一特色在于，它不是由一个方法论构成，而且一个家族。在Alistair Cockburn眼中，项目可以根据参与的人数、影响程度进行划分。

根据项目失败的影响度，可以将其分成4类。

（1）Comfort：如果失败将给人带来不快，也就是影响相对较小的项目。

（2）Discretionary money：如果失败将会带来一定的经济损失的项目。

（3）Essential money：如果失败将会带来巨大的经济损失的项目。

（4）Life：如果失败将会危及生命的项目。

Alistair Cockburn结合自己多年的项目经验，对于各种不同的项目进行总结，提出了水晶方法论族。

Alistair Cockburn借用了不同硬度的水晶表现出来的颜色不同，来区分应用于不同项目的水晶方法，针对更大的项目，他还提出了紫色、蓝色、紫罗兰色等。开发组织可以根据项目实际情况来选择相应的水晶方法。

3．水晶方法论的核心思想

水晶方法的核心价值是以人和交流为中心、高度的宽容性。而且其认为项目必须采用增量式开发，以每 4 个月甚至是更短的时间为一个周期，最好是每 1～3 个月为周期，而且在每个增量周期前、后召开反思讨论会。

4．透明水晶方法

透明水晶是水晶方法中最基本的一个方法，适用于 C6～E6 级的项目。由于参与的人数有限，可以做到（也应该做到）项目只包括一个坐在同一间办公室里的团队。

透明水晶方法定义了 4 种角色，分别是投资人、资深的设计/编码人员、设计/编码人员以及用户。其建议在其中指派一个人充当项目协调人员，而且应该包括业务专家，并且认为资深的设计/编码人员是最关键的人物。

透明水晶方法的核心策略如下：

（1）采用增量式开发，每 2～3 个月定期交付。

（2）用项目的中间成果（可执行软件）为决策、里程碑管理的标志，而非项目文档。

（3）引入部分自动化回归测试系统来保障应用程序的功能。

（4）一定要有用户直接参与。

（5）每次发布前要进行两次用户演示。

（6）当前阶段工作已经稳定到能够评审，就继续下阶段工作。

（7）产品和方法论调整研讨会在每次迭代的开始或中期进行。

（8）所有的这些策略必须全套执行，不可从中吸纳替代做法。

Alistair Cockburn 认为，透明水晶是迄今为止最高宽容度、低正规度、适用于小团队、而且可以工作的方法论。

5．橙色水晶方法论

橙色水晶方法论适合于开发商业化、中等规模的产品项目，通常团队人员总数为 10～40 人，开发周期为 1～2 年，产品的交付时间十分重要，而且不是一失败就会危及生命的项目。

在橙色水晶方法论中，定义了投资人、业务专家、应用专家、技术推动者、业务分析与设计人员、项目经理、架构师、设计顾问、起领导作用的设计/编程人员、设计/编码人员、用户界面设计人员、重用执行者、文档编写人员、测试人员等 14 种角色。

橙色水晶方法认为，应该将较大的功能团队分成多个功能小组，如系统规划、项目监控、系统架构、技术、功能、基础结构、外部测试等部分。

希赛教育专家提示：从这里可以看出，橙色方法论已经明显比较“重”了，但是，与应用于几十人的团队的其他方法相比，其仍然是敏捷的。

18.3.2　动态系统开发方法

动态系统开发方法（Dynamic Systems Development Method，DSDM）是在 20 世纪 90 年代早中期提出的一种形式化 RAD 方法。在长期的实践中，DSDM 又有了新的含义：Dynamic Solution Delivery Model。其中，Dynamic 反映了适应变化的能力；用 Solution

代替 Systems，表示 DSDM 注重客户的解决方案和业务价值；用 Delivery 代替 Development，是将其提升到一个更广阔的高度，表示产品交付性而非传统的任务；而 Model 则比 Method 更能够反映项目中的业务观点。

1．九大原则

DSDM 是一个探索式开发方法，它推崇在软件开发过程中应用著名的 80-20 法则，强调没有什么事能够一次就做好，并将其思想总结成为 9 个原则：

（1）积极的用户参与是必要的。

（2）必须赋予 DSDM 团队决策权。

（3）重点在于产品的经常性交付。

（4）适应业务需要是所交付产品被接受的一个基本标准。

（5）迭代和增量式开发对于最终给出精确的业务解决方案是必要的。

（6）开发期间的任何修改都是可逆的。

（7）需求必须定位在高水平。

（8）测试必须贯穿整个软件生命周期中。

（9）所有项目干系人之间的协作与交流是至关重要的。

2．DSDM 过程概述

DSDM 将软件开发的过程分成了功能模型、设计与构建、实现三个迭代过程。功能模型迭代是一个搜集与确定功能需求的过程，主要通过需求优先级列表来进行；设计和构建迭代则是对功能模型进行细化，生成一个满足所有需求并使设计的软件也满足的原型；然后，再实现迭代完成系统的开发。整个过程如图 18-1 所示。

3．文档与角色

DSDM 对于文档也给予了高度的重视，它定义了 15 个中间工件，但是却没有提供详细的文档模板，仅是提供了一个指导方针、一个简要的描述、一个目标列表以及一些相关的质量标准，而具体的细节留给应用 DSDM 过程的开发团队去补充。

DSDM 定义了 11 种角色，而且还都冠以诸如“空想家”、“大使”这样诙谐的称谓，为其敏捷的文化更是增加了一些趣味性。

18.3.3 特征驱动开发

特征驱动开发的诞生极具有戏剧性。当年，Jeff De Luca 受命于危难，接手了一个新加坡的大型软件开发项目。该项目规模宏大，跨越了三个不同的商业领域，还涉及到对原有系统的集成，而且，这个项目已经有过一次失败的尝试。Jeff De Luca 认为，项目的成功关键点在于领域建模，因此，邀请了他的好友、对象建模专家 Peter Coad 担任项目的主设计师。而该项目正是在这两位黄金拍挡的默契配合之下，取得了卓越的成功。后来，他们将其中的项目经验总结出来，提出了特征驱动开发（Feature-Driven Development，FDD）方法论。

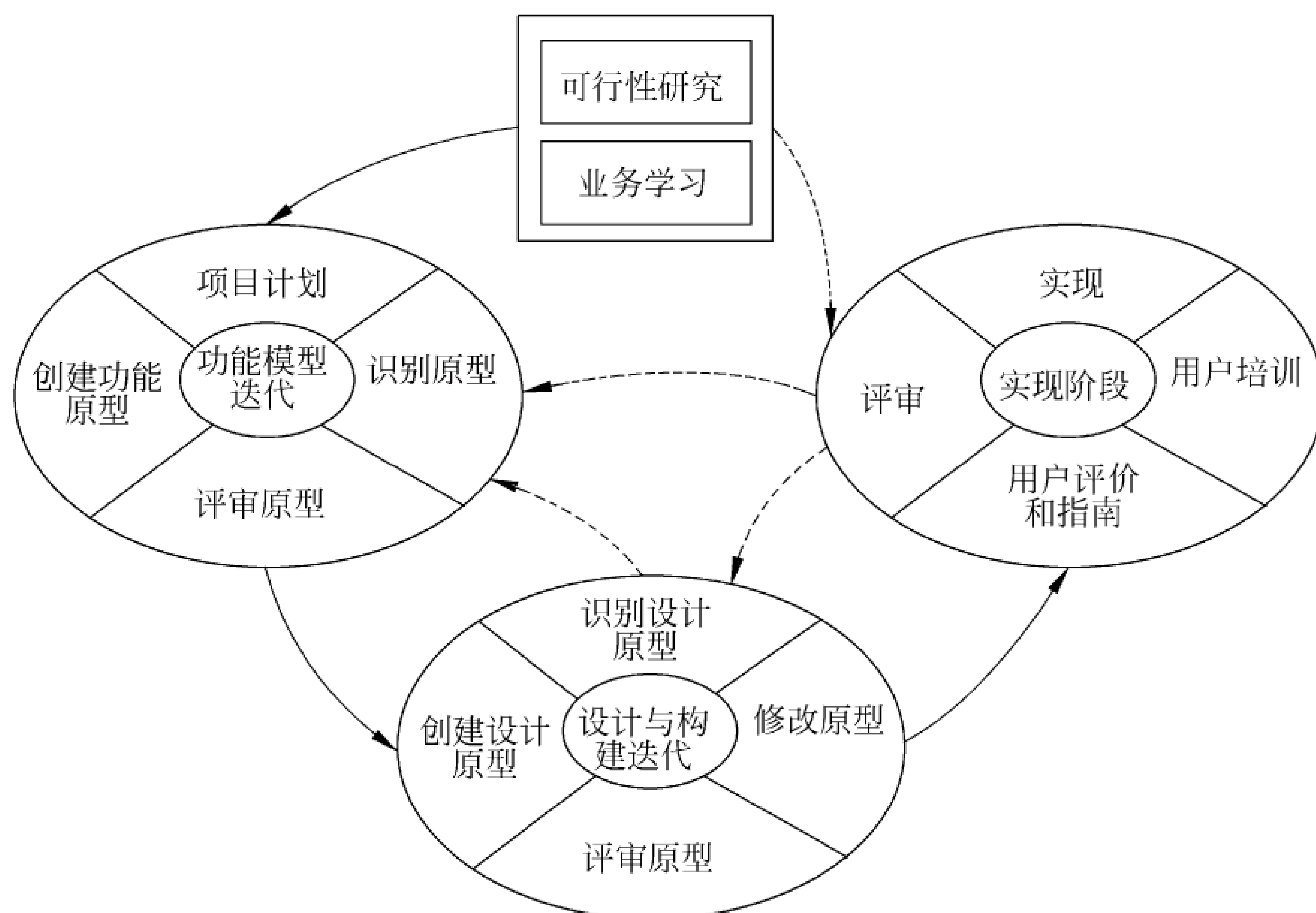


图 18-1　DSDM 过程示意图

1．FDD 的优势

FDD 方法论的最大特点是提倡可高度迭代的软件开发过程，对每一次迭代强调质量，不断地交付切实可行的结果，追求在开销最少、开发人员受干扰最少的情况下，提供精确、有用的进度和状态信息。它能够获得各种不同参与者的认同：

（1）客户：在应用 FDD 方法的软件开发项目中，客户可以不断地获得可执行的中间结果，能够获得直观易懂的进度报告，参与感更强。

（2）高层经理：FDD 所采用的直观易懂的进度报告，可以使得他们更加有效地、正确地控制项目进度，理解团队。

（3）开发人员：FDD 将开发分解成为一个个“特征”，组织一次次的小迭代完成这些特征，每一个特征完成之后，都可以看到实际的效果，感受到软件在一天天的长大。这种过程中不断的完成的感觉，很容易给他们带去成就感、快乐感。

2．FDD 的核心

特征是 FDD 的核心概念，是具有客户价值的功能。这恰好与现代软件开发方法论中提出用例、用户故事两个重要的需求技术的思想高度一致，即都是试图从客户的视角、从问题域的视角来理解和定义软件的需求。这也许是一种巧合，但更是一种必然。

在 FDD 中，采用“<action> <result> <object>”的形式来描述特征。例如，统计本月业务总额（统计是 action，本月业务是 object，总额是 result）。从这个例子中就会发现，特征将会把软件开发团队的视角从技术领域拉出来，转而放到客户价值的体现上。

3．FDD 过程概述

与其他方法论不同的是，FDD 主要致力于软件系统的实际设计和构造，它仅仅关注进行设计和编写软件所需的特定活动和输出。整个 FDD 开发过程从与领域专家合作创建一个领域对象模型开始；然后，结合需求过程中的信息，为开发人员创建一个特征表；接着，在特征表的基础上制定一个迭代的开发计划；最后，通过几个设计、构造的迭代完成开发任务。其整个过程如图 18-2 所示。

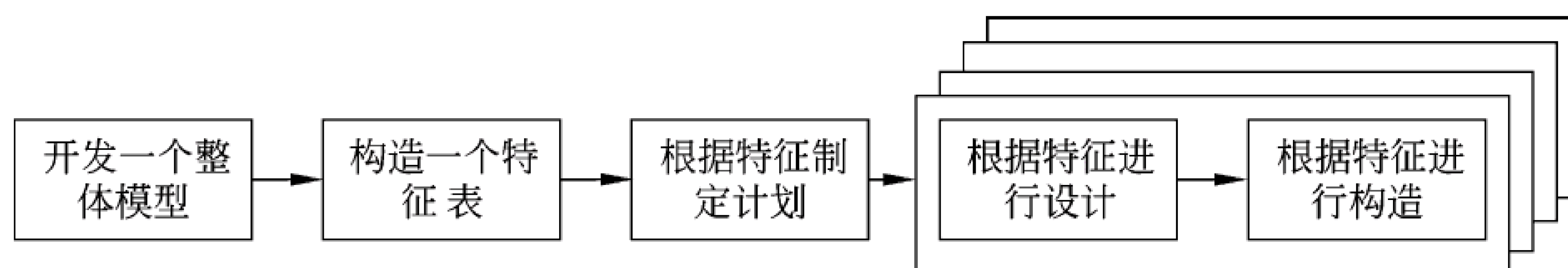


图 18-2 FDD 过程示意图

4．8 个最佳实践

就像很多方法论一样，FDD 也集成了一系列符合其价值理念的最佳实践集合，通过在软件开发过程中组合应用它们，从而贯彻 FDD 的思想。

（1）领域对象建模：FDD 十分强调领域模型的重要性，认为只有这样才能使软件开发中将解决域与问题域结合起来。可以使用类图、E-R 图等方法来建模，当然最好的方法，是使用与 FDD 同时诞生的彩色建模法来完成这一任务。

（2）根据特征组织开发：在整个开发过程中，使用特征为主线进行任务分工与实现、进度监控等工作。这个实践与 UP 中的用例驱动的思想是一致的。

（3）私有的代码所有权：FDD 提倡每个类都有一个责任人，类代码是私有的。从而实现类之间的职责与交互与开发者之间的沟通与交流有效地对应起来，使得软件开发人员融入开发的系统中去。

（4）特征开发小组：在 FDD 中，将形成动态矩阵式管理。一方面是从解决领域着手，每个类都有一个责任人；另一方面则是从问题域着手，每个特征也有一个责任人。将相关的特征组织成为特征集，建立特征开发小组，承担特征、特征集开发的责任。这样，就将系统的组织结构与开发团队的组织结构有机地融合在一起了。

（5）审查：FDD 强调每个环节的质量保证，而保障的方法就是代码审查、设计审查、同级评审等机制。审查的目的是质量控制，而不是绩效考核。

（6）定期构建：FDD 是一个迭代的过程，在开发中将特征分配在几个迭代过程中去，每个迭代都将产生一个可执行的中间成品，这为定期构建提供了可能，也使得在开发的过程中能够定期构建出一些可检验工作的成果。

（7）配置管理：在 FDD 中，强调了配置管理的重要性。这是由于在 FDD 的多次迭代过程中将产生众多的中间版本，没有有力的配置管理做支持是不可想象的。

（8）可视的结果报告：定期构建可以使开发过程能够不断地产生可视的中间产品，

这使得进度更加容易得以监控。另一方面，FDD 还创造了一种可视化的项目进度报表格式，使得进度状态一目了然，十分方便。

18.3.4　自适应软件开发

自适应软件开发（Adaptive Software Development，ASD）是 Jim Highsmith 在从事短间隔、迭代的 RAD 过程开发工作实践中总结而来的。它的精髓在于将复杂自适应系统的思想应用于软件开发领域，复杂度理论可以帮助开发人员理解不可预测性。ASD 的核心思想就是拥抱变化，而非抵制变化，这也是完全符合敏捷思想的。

1．ASD 独特的生命周期模型

ASD 的生命周期模型如图 18-3 所示。

从图 18-3 中可以看出，在 ASD 中，将生命周期归结到三个概念性构件。

（1）推测：即去探索，更清楚地认识到目前所不能够确认的，并且应该大胆地偏离规划。也就是说，应该保持频繁的迭代交付。

（2）协作：ASD 认为复杂的应用程序不是一步构建出来的，而是逐渐演化而来的。而相对于现在变化迅速的时代，没有一个人或一个小组可能了解所有的技术和业务知识，因此，通过联合工作、共享知识来协作就显得十分重要。

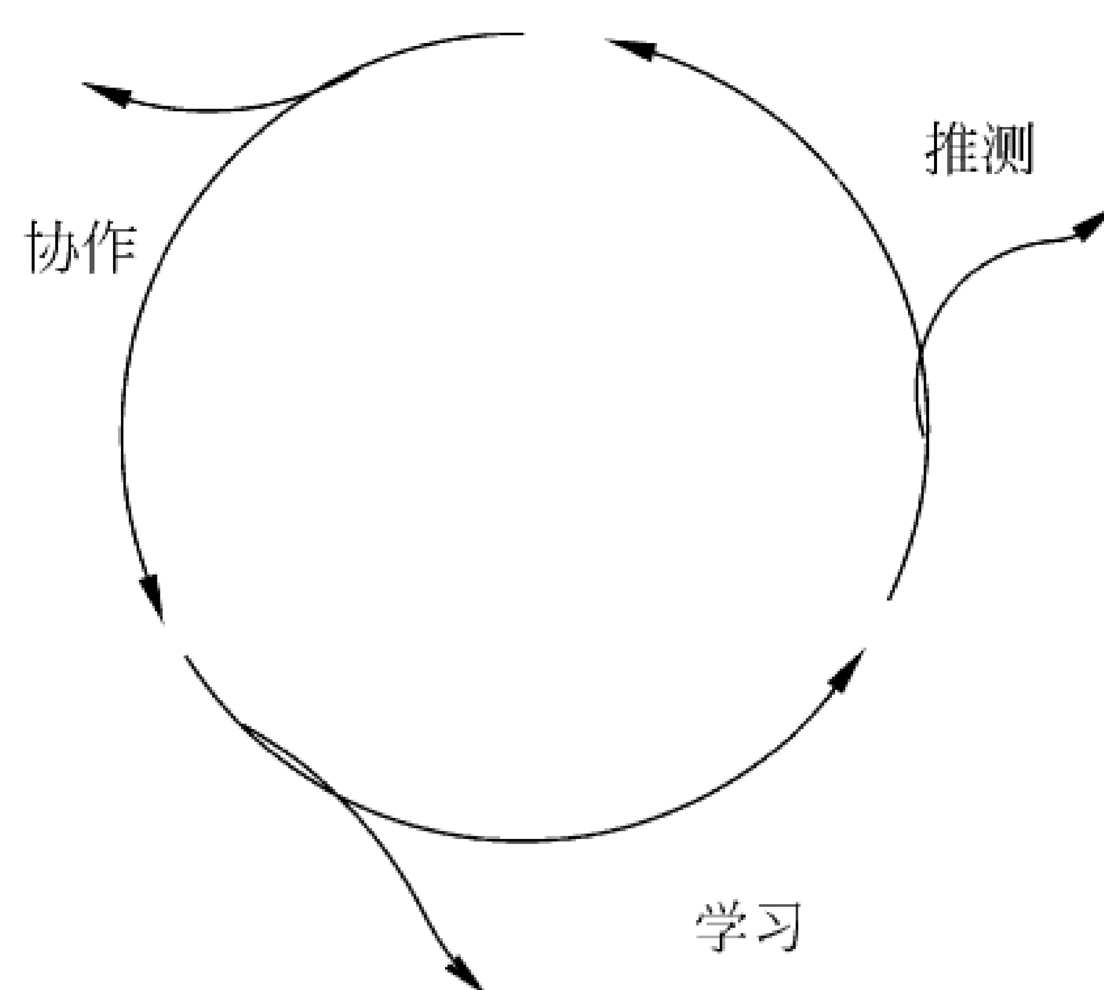


图 18-3　ASD 的生命周期模型

（3）学习：由于每个人都会犯错误，因此，学习对于成功而言，有着更加重要的意义。

希赛教育专家提示：ASD 的生命周期模型与 Barry Boehm 的螺旋开发模型有相似之处，只不过与螺旋模型相比，ASD 更加拥抱变化。

2．ASD 过程概述

ASD 开发过程如图 18-4 所示。

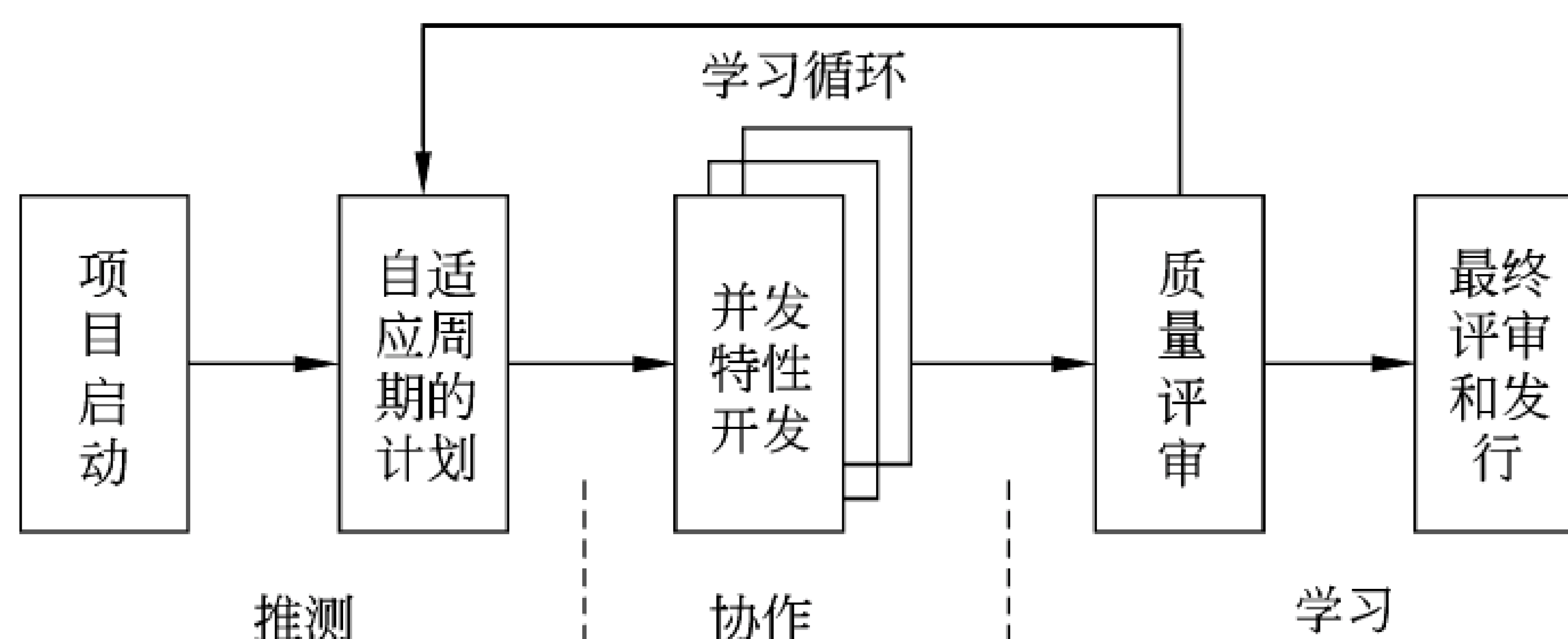


图 18-4　ASD 开发过程示意图

（1）项目启动：包括设置项目的任务和目标，理解约束、建立团队、捕获和分析需求、制定启动规模和范围的推测、确认关键项目风险。

（2）自适应周期的计划：根据项目的范围和特性设置需求，估算并根据项目启动阶段中确定的可用资源确定项目的时间框，最后为时间框分配特性。

（3）并发特性开发：在这个过程中，开发团队负责交付工作软件，项目经理负责协调和并发开发活动。

（4）质量评审：在每次并发的特性开发活动结束之后，应该从客户期待的结果质量、从技术角度期待的结果质量、开发团队自身的过程性能、项目的状态四个角度进行质量评审。

（5）最终评审和发行：当所有的特性开发完成，并通过了所有的质量评审，就可以最终发行了。

3．ASD 眼中的软件开发过程

ASD 将软件开发工作类比成攀岩运动，需要根据实际的情况动态地调整下一个落脚点。在这个过程中，最重要的是根据路途的实际情况进行学习，找到合适的策略与方法。

希赛教育专家提示：ASD 在容变方面已经到达了一种全新的高度，十分适合于电子商务系统等变化极大的项目。

18.3.5 Scrum 方法

Scrum 的创始人 Ken Schwaber 认为，大多数开发的哲学基础都存在错误，因此软件开发方法论不应该是一个“已定义的过程”，而应该是一个“经验式的过程”。Ken Schwaber 认为，由于软件开发工作将会面临巨大的变化，根本不可能预计能够交付什么、需要多少时间、需要花费多少代价。因此，与其使用预先定义好的过程规范，还不如使用持续的反馈机制来管理过程更加有效。

与其他敏捷方法不同的是，Scrum 更突出项目管理方面的问题。Scrum 定义了一个 Sprint 框架过程，包括需求捕获、分析、设计、编程等开发活动。每个 Sprint 是一个 30 天的开发周期，由 Pre-Sprint、Sprint、Post-Sprint 三部分组成。

1．Pre-Sprint

应用 Scrum 过程框架的项目团队，首先应使用一系列的待交付表来记录产品将提供的特性（包括全部的业务和技术特性的列表），当然，这些列表中记录的信息不是一次就完整包括所有的，有的可以在本次 Sprint 完成之后再进行补充。

在 Pre-Sprint 阶段，应该召开由客户、开发人员、项目经理一起参加的 Sprint 技术会议，在会议上根据特性的优先级、复杂度、风险度等各方面的因素，明确地界定本次 Sprint 将要完成的目标。从产品待交付表中找出一些将在本 Sprint 迭代完成的特性，即生成 Sprint 待交付表。

从上面的描述中可以发现，Pre-Sprint 阶段其实就是一个项目计划的阶段，明确目标，

制订计划。

2．Sprint

在 Scrum 过程框架中规定，每次 Sprint 迭代周期是 30 天。在这个 30 天里，团队成员接受任务，每个人都为实现 Sprint 目标努力工作，并通过每日的 Scrum 会议来进行团队协调。Scrum 规定，在 Sprint 迭代中，如果没有极特殊的状况发生，则不允许改变 Sprint 待交付表的内容，也就是说，在这 30 天内，大家要完成的目标不允许更改。这也体现了 Scrum 面对变化的独特构思（即局部稳定）。

在这 30 天的开发周期中，大家并不制订详尽的计划，而是通过团队的天赋与协作来完成任务。因此，为了能够有效地进行协作与沟通，Scrum 建议开发团队每日召开例会。不过，这种 Scrum 会议与传统会议有着很大的不同，其具有以下特点：

（1）相同的时间、不变的地点。

（2）通常维持在 15 分钟之内，最长也不超过半个小时。

（3）团队所有人员都参加，不管是开发还是测试，甚至包括现场客户。

（4）参与的管理人员只听不说。

（5）只提出难点与阻碍，不研究解决方案，留待会后解决。

（6）希望每个与会人员讲述“昨天做了什么、今天打算什么、有什么障碍”。

希赛教育专家提示：敏捷编程方法所提倡的每日短会与 Scrum 会议十分类似，这也是高效会议的好办法，敏捷编程方法为了达到缩短时间的目标，建议站立式召开会议。

3．Post-Sprint

当一个 Sprint 迭代结束之后，就需要召开一个 Post-Sprint 会议来回顾工作的过程，并向客户演示已完成的特性。将客户做出的反馈融合到产品待交付列表中，并为下一次 Sprint 计划会议做好准备。

18.4　极限编程

极限编程（eXtreme Programming，XP）方法可以说是敏捷联盟中最鲜艳的一面旗帜，也是相对来说最成熟的一种。XP 方法的雏形最初形成于 1996—1999 年间，Kent Beck、Ward Cunningham、Ron Jeffery 夫妇在开发 C3 项目（Chrysler Comprehensive Compensation，CCC）的实践中总结出了 XP 的基本元素。在此之后，Kent Beck 和他的一些好朋友们一起在实践中完善提高，终于形成了极限编程方法。

XP 是一种轻量（敏捷）、高效、低风险、柔性、可预测、科学而且充满乐趣的软件开发方式。与其他方法论相比，其最大的不同在于：

（1）在更短的周期内，更早地提供具体、持续的反馈信息。

（2）迭代地进行计划编制，首先在最开始迅速生成一个总体计划，然后在整个项目开发过程中不断地发展它。

（3）依赖于自动测试程序来监控开发进度，并及早地捕获缺陷。

（4）依赖于口头交流、测试和源程序进行沟通。

（5）倡导持续的演化式的设计。

（6）依赖于开发团队内部的紧密协作。

（7）尽可能达到程序员短期利益和项目长期利益的平衡。

XP 由价值观、原则、实践和行为 4 个部分组成，它们彼此相互依赖、关联，并通过行为贯穿于整个生命周期。

18.4.1 四大价值观

XP 的核心是其总结的沟通、简单、反馈、勇气四大价值观，它们是 XP 的基础，也是 XP 的灵魂。

1．沟通

通常，程序员给人留下的印象就是“内向、不善言谈”，项目中的许多问题就出在这些缺乏沟通的开发人员身上。由于某个程序员做出了一个设计决定，但是却不能够及时地通知团队中的其他成员，结果使得团队在协作与配合上出现很多麻烦。而在传统的开发方法中，并不在意这种口头沟通不畅的问题，而是希望借助于完善的流程和面面俱到的文档、报表、计划来替代，但是，这同时又引入了效率不高的新问题。

XP 方法认为，如果小组成员之间无法做到持续的、无间断的交流，那么协作就无从谈起。从这个角度来看，通过文档、报表等人工制品进行交流，具有很大的局限性。因此，XP 组合了诸如结对编程这样的最佳实践，鼓励大家进行口头交流、通过交流解决问题，提高效率。

2．简单

XP 方法提倡在工作中秉承“够用即好”的思路，也就是尽量地简单化，只要今天够用就行，不考虑明天会发现的新问题。这一点看上去十分容易，但要真正做到保持简单的工作其实是很难的，因为在传统的开发方法中，都要求开发人员对未来做一些预先规划，以便对今后可能发生的变化预留一些扩展的空间。

正如传统开发方法的认识一样，许多开发人员也会质疑 XP。保持系统的扩展性很重要，如果都保持简单，那么，如何使得系统能够有良好的扩展性呢？其实不然，保持简单的理由有以下两个：

（1）开发小组在开发时所做的规划，并无法保证其是符合客户需要的，因此，所做的大部分工作都将落空，使得开发过程中重复的、没有必要的工作增多，导致整体效率降低。

（2）在 XP 中，提倡时刻对代码进行重构，一直保持其良好的结构与可扩展性。也就是说，可扩展性和为明天设计并不是同一个概念，XP 是反对为明天考虑而工作，并不是说代码要失去可扩展性。

简单和沟通之间还有一种相当微妙的互相支持关系。一方面，团队成员之间沟通得越多，就越容易明白哪些工作需要做，哪些工作不需要做；另一方面，系统越简单，需要沟通的内容也就越少，沟通也将更加全面。

3．反馈

是什么原因使得我们的客户、管理层这么不理解开发团队？究其症结，就是开发的过程中缺乏必要的反馈。在很多项目中，当开发团队经历过了需求分析阶段之后，在相当长的一个时间段中，是没有任何反馈信息的。整个开发过程对于客户和管理层而言就像一个黑盒子，进度完全不可见。而且，在项目开发过程中，这样的现象不仅出现在开发团队与客户、管理层之间，还包括在开发团队内部。因此，开发团队需要更加注重反馈。反馈对于任何软件项目的成功都是至关重要的，而在 XP 方法论中则更进一步，通过持续、明确的反馈来暴露软件状态的问题。具体而言就是：

（1）在开发团队内部，通过提前编写单元测试代码，及时反馈代码的问题与进展。

（2）在开发过程中，还应该加强集成工作，做到持续集成，使得每一次增量都是一个可执行的工作版本，也就是逐渐使软件长大。整个过程中，应该通过向客户和管理层演示这些可运行的版本，以使得及早地反馈，及早地发现问题。

反馈与沟通有着良好的配合，及时和良好的反馈有助于沟通。而简单的系统，更有利于测试和反馈。

4．勇气

在应用 XP 方法时，每时每刻都在应对变化：由于沟通良好，会有更多需求变更的机会；由于时刻保持系统的简单，新的变化会带来一些重新开发的需要；由于反馈及时，会有更多中间打断思路的新需求。总之，这一切使得开发团队处于变化之中，因此，这时就需要有勇气来面对快速开发，面对可能的重新开发。

XP 方法要求开发人员穿上强大、自动测试的“盔甲”，勇往直前，在重构、编码规范的支持下，有目的地快速开发。

勇气可以来源于沟通，因为它使得高风险、高回报的试验成为可能；勇气可以来源于简单，因为面对简单的系统，更容易鼓起勇气；勇气可以来源于反馈，因为可以及时获得每一步前进的状态（自动测试），会使得更勇于重构代码。

希赛教育专家提示：在 XP 的四大价值观之下，隐藏着一种更深刻的东西，那就是尊重。因为这一切都建立在团队成员之间的相互关心、相互理解的基础之上。

18.4.2　十二个最佳实践

在 XP 中，集成了 12 个最佳实践，有趣的是，它们没有一个是创新的概念，大多数概念和编程一样老。其主要的创新点在于提供一种良好的思路将这些最佳实践结合在一起，并且确保尽可能彻底地执行它们，使得它们能够在最大程度上互相支持。

1．计划游戏

计划游戏的主要思想就是先快速地制定一份概要的计划，然后，随着项目细节的不断清晰，再逐步完善这份计划。计划游戏产生的结果是一套用户故事以及后续的一两次迭代的概要计划。

“客户负责业务决策，开发团队负责技术决策”是计划游戏获得成功的前提条件。也就是说，系统的范围、下一次迭代的发布时间、用户故事的优先级应该由客户决定，而每个用户故事所需的开发时间、不同可选技术的成本、如何组建团队、每个用户故事的风险以及具体的开发顺序应该由开发团队决定。

客户和开发人员坐在同一间屋子里，每人都准备一支笔、一些用于记录用户故事的纸片，最好再准备一个白板，就可以开始“游戏”了。

（1）客户编写故事：首先由客户谈论系统应该完成什么功能，然后用通俗的自然语言，使用自己的词汇，将其写在卡片上，这就是所谓的用户故事。

（2）开发人员进行估算：首先，客户按优先级将用户故事分成必须要有、希望有、如果有更好三类；然后，开发人员对每个用户故事进行估算，先从高优先级的开始估算。如果在估算的时候，感到有一些故事太大，不容易进行估算，或是估算的结果超过 2 人周，那么就应该对其进行分解，拆成 2 个或多个小故事。

（3）确定迭代的周期：接下来就是确定本次迭代的时间周期，这可以根据实际的情况进行确定，不过最佳的迭代周期是 2～3 周。有了迭代的时间之后，再结合参与的开发人数，算出可以完成的工作总量。然后，根据估算的结果，与客户协商，挑出时间上足够、优先级合适的用户故事组合，形成计划。

2．小型发布

XP 方法秉承的是“持续集成、小步快走”的哲学，也就是说每一次发布的版本应该尽可能地小，当然前提条件是每个版本有足够的商业价值，值得发布。

由于小型发布可以使得集成更频繁，客户获得的中间结果越频繁，反馈也就越频繁，客户就能够实时地了解项目的进展情况，从而提出更多的意见，以便在下一次迭代中计划进去，以实现更高的客户满意度。

3．隐喻

相对而言，隐喻比较令人费解。根据词典中的解释是：“一种语言的表达手段，它用来暗示字面意义不相似的事物之间的相似之处”。隐喻常用于 4 个方面。

（1）寻求共识：鼓励开发人员在寻求问题共识时，可以借用一些双方都比较熟悉的事物来做类比，从而帮助大家更好地理解解决方案的关键结构，也就是更好地理解系统是什么、能做什么。

（2）发明共享语汇：通过隐喻，有助于提出一个用来表示对象、对象间的关系的通用名称，如策略模式（用来表示可以实现多种不同策略的设计模式）、工厂模式（用来表示可以按需“生产”出所需的类的设计模式）等。

（3）创新的武器：有时候，可以借助其他东西来找到解决问题的新途径。例如，可以将工作流看作是一条生产线。

（4）描述架构：架构是比较抽象的，引入隐喻能够大大减轻理解的复杂度。例如，管道架构就是指两个构件之间通过一条传递消息的管道进行通信。

希赛教育专家提示：如果能够找到合适的隐喻是十分快乐的，但并不是每一种情况都可以找到恰当的隐喻，因此，没有必要去强求，而是顺其自然。

4．简单设计

强调简单的价值观，引出了简单性假设原则，落到实处就是“简单设计”实践。这个实践看上去似乎很容易理解，但却又经常被误解，许多批评者就指责 XP 忽略设计是不正确的。其实，XP 的简单设计实践并不是要忽略设计，而且认为设计不应该在编码之前一次性完成，因为那样只能建立在“情况不会发生变化”或“我们可以预见所有的变化”之类的谎言的基础上的。

Kent Beck 概念中的简单的设计是这样的：能够通过所有的测试程序，没有包括任何重复的代码，清楚地表现出程序员赋予的所有意图，包括尽可能少的类和方法。他认为要想保持设计简单的系统，需要具备简单思考的能力，拥有理解代码和修改代码的勇气，以及为了消除代码“坏味道”而定期重构的习惯。

那么，如何开始进行简单的设计呢？XP 的实践者们也总结出了一些具体的、可操作的思考方法。

（1）首先写测试代码：具体将在后面详细描述。

（2）保持每个类只负责一件事：单一职责原则（Single Responsibility Principle，SRP）是面向对象设计的基本原则之一。

（3）使用迪米特（Demeter）法则：迪米特法则也称为 LoD 法则、最少知识原则。也就是指一个对象应当对其他对象尽可能少的了解。用隐喻的方法来解释的话，就是“只与你直接的朋友通信”、“不要和陌生人说话”。

5．测试先行

当笔者第一次看到“测试先行”这个概念的时候，第一感觉就是不解，陷入了“程序都还没有写出来，测试什么呀？”的迷思。于是，开始天马行空地寻求相关的隐喻，终于找到了能够启发我的泥瓦匠。首先，来看看两个不同泥瓦匠是如何工作的吧。

工匠甲：先拉上一根水平线，砌每一块砖时，都与这根水平线进行比较，使得每一块砖都保持水平。

工匠乙：先将一排砖都砌完，然后再拉上一根水平线，看看哪些砖有问题，对有问题的砖进行适当的调整。

显然，读者都会觉得工匠乙的做法浪费时间。然而仔细想想，自己平时在编写程序时其实就是按照工匠乙的方法的，甚至有时候比工匠乙还笨，是整面墙都砌完了，直接进行“集成测试”，经常让整面的墙倒塌。

对于有些团队而言，没有采用工匠甲的工作方法，甚至有时候程序员会以“开发工作太紧张”为理由，而忽略测试工作。这样，就导致了一个恶性循环，越是没空编写测试程序，代码的效率与质量越差，花在找 Bug、解决 Bug 的时间也越来越多，实际产能大大降低。由于产能降低了，因此时间更紧张，压力更大。

6．重构

重构是一种对代码进行改进而不影响功能实现的技术，XP 需要开发人员在“闻到代码的坏味道”时，有重构代码的勇气。重构的目的是降低变化引发的风险、使得代码优化更加容易。通常重构发生在两种情况之下。

（1）实现某个特性之前：尝试改变现有的代码结构，以使得实现新的特性更加简单。

（2）实现某个特性之后：检查刚刚写完的代码后，认真检查一下，看是否能够进行简化。

在考虑重构时，应该要养成编写并经常运行测试代码的习惯；要先编写代码，再进行重构；把每一次增加功能都当作一次重构的好时机；将每一个纠正错误当作一次重构的时机。重构技术是对简单设计的一个良好的补充，也是 XP 中重视“优质工作”的体现，这也是优秀的程序员必备的一项技能。

7．结对编程

自从 20 世纪 60 年代开始，就有类似的实践在进行，长年以来的研究结果给出的结论是，结对编程的效率反而比单独编程更高。一开始虽然会牺牲一些速度，但慢慢地，开发速度会逐渐加快。究其原因，主要是结对编程大大降低了沟通的成本，提高了工作的质量。具体表现在以下几个方面：

（1）所有的设计决策确保了不是由一个人做出的。

（2）系统的任何一个部分都肯定至少有两个人以上熟悉。

（3）几乎不可能有两个人都忽略的测试项或其他任务。

（4）结对组合的动态性，是一个组织进行知识管理的好途径。

（5）代码总是能够保障被评审过。

而且，XP 方法集成的其他最佳实践也能够使得结对编程更加容易进行：

（1）编码标准可以消除一些无谓的分歧。

（2）隐喻可以帮助结对伙伴更好地沟通。

（3）简单设计可以使结对伙伴更了解他们所从事的工作。

结对编程技术被誉于 XP 保持工作质量、强调人文主义的一个最典型的实践，应用得当还能够使开发团队协作更加顺畅、知识交流与共享更加频繁、团队稳定性也会更加牢固。

8．集体代码所有制

由于 XP 方法鼓励团队进行结对编程，而且认为结对编程的组合应该动态的搭配，根据任务的不同、专业技能的不同进行最优组合。因此，每一个人都会遇到不同的代码，

代码的所有制就不再适合于私有，因为那样会给修改工作带来巨大的不便。

所谓集体代码所有制，就是团队中的每个成员都拥有对代码进行改进的权利，每个人都拥有全部代码，也都需要对全部代码负责。同时，XP 强调代码是谁破坏的（修改后出现问题），就应该由谁来修复。

由于在 XP 中，有一些与之相匹配的最佳实践进行配合，因此并不要担心采用集体代码所有制会让代码变得越来越乱。

（1）由于在 XP 项目中，集成是一件经常性的工作，因此，当有人修改了代码带来了集成的问题，会在很快的时间内被发现。

（2）由于每一个类都会有一个测试代码，因此，不论是谁修改了代码，都需要运行这个测试代码，这样，偶然性的破坏发生的概率将很小。

（3）由于每一个代码的修改都通过了结对的两个程序员共同的思考，因此，通常做出的修改都是对系统有益的。

（4）由于大家都坚持了相同的编码标准，因此，代码的可读性、可修改性都会比较好，而且还能够避免由于命名法、缩进等小问题引发经常性的代码修改。

希赛教育专家提示：集体代码所有制是 XP 与其他敏捷方法的一个较大不同，也是从另一个侧面体现了 XP 中蕴含的很深厚的编码情节。

9．持续集成

在前面谈到小型发布、重构、结对编程、集体代码所有制等最佳实践的时候，多次提到“持续集成”，可以说持续集成是这些最佳实践的基本支撑条件。

可能读者会对持续集成与小型发布代表的意思容易混淆不清，其实，小型发布是指在开发周期中经常发布中间版本，而持续集成的含义则是要求 XP 团队每天尽可能多次地做代码集成，每次都在确保系统运行的单元测试通过之后进行。这样，就可以及早地暴露、消除由于重构、集体代码所有制所引入的错误，从而减少解决问题的痛苦。

要在开发过程中做到持续集成并不容易。集成工作往往是十分枯燥、繁琐的，因此，适当地引入每日集成工具是十分必要的。XP 方法建议首先使用配置管理服务器将代码管理起来，然后使用 Ant 或 Nant 等 XP 工具，编写集成脚本，调用 xUnit 等测试框架，这样，就可以实现每当程序员将代码 Check in 到配置服务器上时，Ant 就会自动完成编译和集成，并调用测试代码完成相应的测试工作。

10．每周工作 40 小时

这是最让开发人员开心、管理者反对的一个最佳实践了，加班、再加班早已成为开发人员的家常便饭，也是管理者最常使用的一种策略。而 XP 方法认为，加班最终会扼杀团队的积极性，最终导致项目的失败，这也充分体现了 XP 方法关注人的因素比关注过程的因素更多一些。

Kent Beck 认为，开发人员即使能够工作更长时间，他们也不应该这样做，因为这样做会使他们容易厌倦编程工作，从而产生一些影响他们产能的其他问题。因此，每周工

作 40 小时是一种顺势而为，是一种规律。其实，对于开发人员和管理者来说，违反这种规律都是不值得的。

（1）开发人员：如果不懂得休息，那么就无法将自己的节奏调整到最佳状态，就会带来很大的负面影响。而且，在精神不集中的状态下，开发的质量也得不到保证。

（2）管理者：每个开发人员的工作精力是有限的，不可能无限增长，在精力不足的时候，不仅写出来的代码质量没有保障，而且还可能为项目带来退步的效果。因此，采用加班的方式并不是一个理性的方式，是得不偿失的。

不过，有一点是需要解释的，“每周工作 40 小时”中的“40”不是一个绝对数，它所代表的意思是团队应该保证按照“正常的时间”进行工作。

11．现场客户

为了保证开发出来的结果与客户的预想接近，XP 方法认为最重要的是需要将客户请到开发现场。就像计划游戏中提到过的，在 XP 项目中，应该时刻保证客户负责业务决策，开发团队负责技术决策。因此，在项目中有客户在现场明确用户故事，并做出相应的业务决策，对于 XP 项目而言有着十分重要的意义。

也许有人会问，客户提交了用户故事之后不就完成工作了吗？其实很多尝试过用户故事的团队都会发现其太过简单，包含的信息量极少，XP 方法不会不了解，因此其不会把用户故事当作开发人员交付代码的唯一指示。用户故事只是一个起点，后面的细节还需要开发人员与客户之间建立起来的良好沟通来补充。

作为一名有经验的开发人员，绝对不会对现场客户的价值产生任何怀疑，但是，都会觉得要想实现现场客户十分困难。要实现这一点，需要对客户进行沟通，让其明白，相对于开发团队，项目成功对于客户而言更为重要。而现场客户则是保障项目成功的一个重要措施。

其实，在具体实施中，也不是一定需要客户一直和开发团队在一起，而是开发团队应该和客户能够随时的沟通，可以是面谈，可以是在线聊天，可以是电话，当然面谈是必不可少的。其中的关键是当开发人员需要客户做出业务决策时，需要进一步了解业务细节时，能够随时找到相应的客户。

不过，也有一些项目是可以不要现场客户的参与的。例如，当开发组织中已经有相关的领域专家时；当做一些探索性工作，而且客户也不知道他想要什么时（如新产品、新解决方案的研究与开发等）。

12．编码标准

编码标准是一个“雅俗共享”的最佳实践，不管是代表重型方法论的 UP、PSP，还是代表敏捷方法的 XP，都认为开发团队应该拥有一个编码标准。XP 方法认为拥有编码标准可以避免团队在一些与开发进度无关的枝末细节问题上发生争论，而且会给重构、结对编程带来很大的麻烦。不过，XP 方法的编码标准的目的不是创建一个事无巨细的规则列表，而是只要能够提供一个确保代码清晰，便于交流的指导方针。

如果开发团队已经拥有编码标准，就可以直接使用它，并在过程中进行完善。如果还没有，那么可以先进行编码，然后在过程中逐步总结出编码规则，边做边形成。当然，除了这种文字规范以外，还可以采用一些如自动格式化代码工具之类的方法进行代码规范。事实上，只需要很好地贯彻执行其他实践，并且进行沟通，编码标准会很容易地浮现出来。

13．融合是关键

有句经典名言“1+1>2”最适合表达 XP 的观点，Kent Beck 认为，XP 方法的最大价值在于，在项目中融会贯通地运用这 12 个最佳实践，而非单独使用。当然，可以使用其中的一些实践，但这并不意味着就应用了 XP 方法。XP 方法真正能够发挥其效能，就必须完整地运用 12 个实践。

本章参考文献

[1] Kent Beck. 解析极限编程—拥抱变化. 北京：人民邮电出版社，2002

[2] Kent Beck，Martin Fowler. 规划极限编程. 北京：人民邮电出版社，2002

[3] Ron Jeffries，Ann Anderson，Chet Hendrickson. 极限编程实施. 北京：人民邮电出版社，2002

[4] Giancarlo Succi，Michele Marchesi. 极限编程研究. 北京：人民邮电出版社，2002

[5] James Newkirk，Robert CMartin. 极限编程实践. 北京：人民邮电出版社，2002

[6] William C Wake. 探索极限编程. 北京：人民邮电出版社，2002

[7] Ken Auer，Roy Miller. 应用极限编程—积极求胜. 北京：人民邮电出版社，2002

[8] Robert C Martin. 敏捷软件开发：原则、模式与实践. 北京：清华大学出版社，2003

[9] Alistair Cockburn. 敏捷软件开发. 北京：人民邮电出版社，2003

[10] Stewart Baird. 极限编程—基础、案例与实施. 北京：人民邮电出版社，2003

[11] James A.Highsmith. 自适应软件开发. 北京：清华大学出版社，2003

[12] Stephen R.Palmer，John M.Felsing. 特征驱动开发方法—原理与实践. 北京：机械工业出版社，2003

[13] Jim Highsmith. 敏捷软件开发生态系统. 北京：机械工业出版社，2004

[14] Kent Beck. 测试驱动开发. 北京：中国电力出版社，2003

[15] Martin Fowler. 重构—改善既有代码的设计. 北京：中国电力出版社，2003

第 19 章　P2P 技术

P2P（Peer-to-Peer）的意思主要有“对等者”、“同事”或“伙伴”。因此，P2P 在计算机领域里可称为对等连接或对等网络，它是近几年以来深受 IT 业界关注的一个概念。

P2P 是一种技术，它是基于 P2P 拓扑结构发展起来的一项新型的分布式网络通信技术，它使得计算机之间可以直接访问和交换文件，而不是像过去那样连接到服务器去浏览与下载。P2P 的参与者共享他们所拥有的一部分硬件资源（如处理能力、存储能力、网络连接能力、打印机等），这些共享资源需要由网络提供服务和内容，能被其他 peer 直接访问而无须经过中间实体。在此网络中的参与者既是资源（服务和内容）提供者，又是资源（服务和内容）使用者。

P2P 技术改变了互联网（现在的）以大网站为中心的状态，使得网络资源处于“非中心化”的地位，并把访问权交还给对等的用户。它使得网络沟通变得容易，共享和交互更直接，真正消除了中间商。

19.1　P2P 概述

P2P 不单是一种技术，也代表一种思想和全新的交流方式，它代表整个互联网技术革新的一种开拓创新思想，P2P 的出现改变了人们对 Internet 的理解与认识，它直接将人们联系起来，让人们通过互联网直接交互，因此，P2P 也是一种社会和经济现象。

P2P 之所以广泛吸引人们的注意，主要在于以下几个方面：

（1）高可用的超大规模计算和存储资源共享。

（2）网络联通功能更强大，信息沟通更直接、更灵活。

（3）巨大的扩展力。通过低成本交互来聚合资源，导致整体大于部分之和。

（4）低成本的所有权和共享。使用现存的基础设施、削减和分布成本。

（5）匿名和隐私。允许对等端在其数据和资源上很大的自治控制。

目前，P2P 在加强网络上人的交流、文件共享和交换、分布式计算等方面已经充分显示出了其强大的技术优势。

19.1.1　产生的背景

P2P 技术是在早期的 C/S 模式下变革而来的，也是在 Napster 产生后所发展的必然结果。

1．C/S 模式的变革

在早期的互联网环境下，人们把一些资源存放到功能比较强大的服务器中，其他的客户端通过访问服务器获得信息和资源。这就是一种基于 C/S 结构的网络通信模式。在 C/S 模式中，数据的分发采用专门的服务器，多个客户端都从同一台服务器中获取数据。这种模式的优点是：数据的一致性容易控制，系统也容易管理。但是此种模式的缺点是：因为服务器的台数只是有限的少数几台，要面对众多的客户端，由于 CPU 能力、内存大小、网络带宽的限制，可同时服务的客户端数量非常有限，可扩展性差，服务器的瓶颈容易凸现。因此，系统容易出现单一故障点，一旦服务器出现问题，就会导致整个网络瘫痪，资源无法得到充分利用。例如，地震后数据容灾中心的破坏、战争后指挥中心应用数据的毁灭等，这就是"中心化"结果所带来的弊端。

随着对互联网的不断接触和了解，人们已经清楚地知道，在 Internet 上最大的资源拥有群不是服务器而是客户机，最多最好的资源实际上是存在于每个人的 PC 中。因此，客户机才是 Internet 的主体。随着硬件的不断发展，现在的 PC 可以提供大容量的存储能力和高速的计算能力。人们迫切希望能打破服务器的垄断，在 Internet 上拥有属于自己的空间。同时，为了解决由于中心化所带来的弊端，人们迫切需要把中心化的集中方式转化为非中心化的分散方式，于是，P2P 技术便因此而诞生。

2．Napster 的音乐版权风波对 P2P 技术的推动

1998 年，美国东北波士顿大学的一年级新生肖恩• 范宁，为了能够帮助他的室友在网上找到音乐，而在波士顿的东北大学校园上开发了一个名为 Napster 的小程序，它把所有的音乐文件地址存放在一个集中的服务器中，让使用者通过资源地址而方便地找到自己需要的 MP3 文件，于是，建成了 Napster 在线音乐商店。在这里，Napster 本身并不提供 MP3 文件的下载，它实际上提供的是整个 Napster 网络的 MP3 文件目录，因为 MP3 文件分布在网络中的每一台机器中，随时供用户选择下载使用，用户下载都是直接连到另外一台机器，传输速度也相当快。Napster 具有强大的搜索功能，可以将在线用户的 MP3 音乐信息进行自动搜寻并分类整理，以备其他用户查询，用户只要知道喜欢歌曲的名称或演唱者的名称，就可以选择自己要与其他人在网上共享的音乐文件的目录，与全世界乐迷进行音乐共享，并且可以进行聊天、在线讨论与交流。

Napster 创造了奇迹，同时也揭示了在互联网时代普通人也具有改变整个世界的能力。Napster 不单是实现了 MP3 歌曲文件的共享，更主要的是这个小程序改变了整个世界，它唤醒了深藏在互联网背后的 P2P 对等联网技术，由此促成了这种技术向整个互联网拓展。

19.1.2　研究内容和目标

从应用角度来看，目前，P2P 技术研究主要涉及到以下几个领域：

（1）提供文件和其他内容共享的 P2P 网络，如 Napster、Gnutella、CAN、eDonkey、

BitTorrent 等。

（2）挖掘 P2P 对等计算能力和存储共享能力，如 SETI@home、Avaki、Popular-Power 等。

（3）基于 P2P 方式的协同处理与服务共享平台，如 JXTA、Magi、Groove、.NETMy-Service 等。

（4）即时通信交流，如 ICQ、OICQ、Yahoo Messenger 等。

（5）安全的 P2P 通信与信息共享，如 CliqueNet、Crowds、OnionRouting 等。

P2P 技术的研究目标包括应用和技术两个方面。P2P 技术的应用目标是为了满足应用的需要，主要有信息、服务的共享与管理；协同工作；构建充当基层架构的互联系统。具体包括共享与削减成本、资源聚合与互操作能力、增加自治功能、隐私性和匿名性、动态性（资源动态进入或离开系统）、实现 Ad-hoc 通信和协同。

P2P 的技术目标是尽可能非中心化，减少集中式的服务器方式，即 Serverless 或 Noserver，有两种情况：

（1）少量借助服务器（with-Server）的方式。这种方式的一个主要特点是服务器的功能已经远远退化，一般只作为资源索引服务器（Index Server，IS）使用，提供所有 Peer 以及之上的各种文件列表查找索引服务，是现在比较常见的 P2P 解决方案。这种方式的典型代表有 Napster、eDoney＆eMule、Jelawat、Workslink 等。目前，这类产品多以文件共享服务为主，并兼有简单的即时通信功能。

（2）完全脱离服务器的方式（non-Server）。这种方式下所有 Peer 都是平等的，完全不需要服务器的存在，在 P2P 网络中所有的资源按照某种规则共享，同时任何 Peer 可以在任何时候、任何地点加入到某个 P2P 网络群体中，不需要服务器的配合和支持。这是 P2P 研究的重点和难点，也是 P2P 技术的最终目标。

19.1.3 对互联网的影响

P2P 技术挖掘了互联网巨大的潜力，将互联网从一个基于文件的网页和电子邮件网络转变成一个动态的、颗粒状网络，在网络中，特定的信息可被有效地放置和分享。具体来说，P2P 技术对互联网带来的影响主要表现在以下 4 个方面。

首先，P2P 技术改变互联网中以服务器资源集中为依托的运作管理模式，实现了“网络就是计算机，计算机就是网络”的梦想。P2P 技术的出现与发展，让人们重新发现了一种全新的文件交换方式。长久以来，人们已经习惯了互联网的 C/S 中心服务模式，就是以服务器为中心，人们向服务器发送请求，然后服务器回应给个人需要的信息。但网络的特点，应该是以用户为中心，所有的用户都是平等的伙伴。远程用户可以通过直接互联而共享硬盘上的文件、目录乃至整个硬盘。所有人都可以共享他们认为最有价值的东西，这将使互联网上信息的价值得到极大的提升。同时，用户之间直接交流的方式使人们对于开放的、自由的互联网的理想变为现实。

其次，P2P 技术扩宽了互联网的应用范围，扩大了人们的网络生活空间。P2P 技术的推广使得很多应用被逐步开发出来，对等计算、协同工作、搜索引擎、文件交换、多媒体点播、网络直播、网络电视等已经扩宽了互联网的应用范围，改变了人们的网络生活，也给人们带来了新的商机。同时，网民的上网习惯、企业的运作方式、相关法律等诸多方面都在受到 P2P 的冲击。因此，P2P 扩大了人们的网络生活空间，使人们能够重新认识和重新参与互联网。

再次，P2P 技术促成了互联网、电信网和广电网三网之间的日趋融合。P2P 技术的影响力不仅局限于互联网，还间接地延伸到了电信网和广电网。使用 P2P 技术结合跨越互联网和电信网的基础性架构的 ENUM（电话号码映射）技术协助解决了三网融合的技术瓶颈，可以使 P2P 共享的优势延伸到电信网络，加速互联网、电信网和广电网的融合，以 Skype、BT、PPLlive 为代表的 P2P 语音、P2P 流媒体、IPTV 等业务的迅速发展挑战了电信网、广电网在语音服务、视频传播方面的统治地位，间接地促使电信、广电网力求转变运营模式，向三网融合的方向发展。三网融合可以更加有效提高网络资源的利用水平、为用户提供更丰富便捷的服务，以及较高水平的网络性能，将更有利于实现用户利益的最大化。如今，三网融合将在 P2P 所带来的“人人为我、我为人人”的无限分享理念的促使下逐渐发展，也成为很多国家致力于信息化建设和发展的目标。

最后，P2P 技术改变了互联网的模式和流量。P2P 技术将带来一个业务上“全分布”式的网络。流量将呈现出更大的任意性，用户之间直接的数据交换将更加频繁。P2P 技术在应用层的组网，在为网络应用的运营者带来更大的灵活性的同时，也造成了基础承载网络资源紧张，网络设备长时间处于满负荷工作状态。P2P 对网络模型的影响主要体现在以下几个方面：

（1）由于 P2P 对称特点和 P2P 流量比例增加，城域网的流量模型逐渐从不对称迁移到对称，与接入网 xDSL 不对称网络形成明显的矛盾。

（2）P2P 的流向处于一种无序的状态，造成网络性能质量劣化和拥塞，带宽大量消耗而收益为零。当前大多数 P2P 工具为了保证传输质量，往往创建大量连接而并未传输数据，消耗了网络资源。

（3）跨区流量大于区内的流量，造成主干出口扩容压力不断增大。

19.1.4　需要解决的关键问题

P2P 的计算模式需要解决资源放置、资源定位、资源获取等三个问题。

1．资源放置

在 P2P 系统中，并非个人资源都放置在各自的机器上，很可能是所有机器共同管理资源。例如，在 P2P 存储系统中，经常采用分布式哈希表放置数据，某个用户的数据可能放置在其他人的机器上。于是，如何进行资源放置就成了必须回答的第一个问题。

2．资源定位

数据的查找与资源放置方法是直接相关的。对于以分布哈希表（Distributed Hash

Table，DHT）方式放置的数据，可以直接定位，但在多数文件共享系统中，用户的文件都是放在各自的机器上，如何知道哪些机器放有用户需要的数据就成为一个关键问题，常常需要较大规模的搜索才可以完成。资源定位就是研究如何更有效地找到需要的资源所处的位置，尤其是一些在网络中稀有的数据。

在 P2P 网络中进行资源定位一般采用以下三种方式：

（1）集中方式索引。每一个节点将自身能够提供共享的内容注册到一个或几个集中式的目录服务器中。查找资源时首先通过服务器定位，然后，两个节点之间再直接通信。例如，早期的 Napster 就是使用这种方式。这种方式的网络实现简单，但往往需要大的目录服务器的支持，并且系统的健壮性不好。

（2）广播方式。没有任何索引信息，内容提交与查找都通过相邻接节点直接广播传递。例如，Gnutella 就是使用这种方式。一般情况下，采取这种方式的 P2P 网络对参与节点的带宽要求比较高。

（3）分布哈希表的方式。DHT 是大多数 P2P 网络所采取的资源定位方式。首先将网络中的每一个节点分配虚拟地址（vid），同时用一个关键字（key）来表示其可提供的共享内容，取一个哈希函数 H，这个函数可以将 key 转换成一个哈希值 H(key)，网络中节点相邻的哈希值相邻，发布信息时把（key，vid）二元组发布到具有和 H(key)相近地址的节点上去，其中 vid 指出了文档的存储位置，在进行资源定位的时候，就可以快速根据 H(key)到相近的节点上获取二元组（key，vid），从而获得文档的存储位置。不同的 DHT 算法决定了 P2P 网络的逻辑拓扑，比如 CAN 就是一个 N 维向量空间，而 CHORD 是一个环形拓扑，TAPESTRY 则是一个网状的拓扑。

上述的资源定位方式可以依据不同的 P2P 应用环境进行选择，但是，人们普遍看好 DHT 方法。基于 DHT 的 P2P 网络在一定程度上可以直接实现内容的定位。但有一个问题是，如果一个节点提供共享的内容表示越复杂，则哈希函数越不好选择，相应地，网络的拓扑结构就越复杂；而如果内容表示简单，则又达不到真正实现依据内容定位的能力。目前，大多数 DHT 方式的 P2P 网络对节点所提供共享内容的表示都很简单，一般仅仅为文件名。

3．资源获取

资源定位后就需要获得资源，有些资源并不能直接获得，如计算资源、大文件、流媒体资源等。问题主要在于如何才能更高效地获取资源，或说如何使一些热点资源服务更多的需要该资源的用户，通常这需要尽量发挥 P2P 系统中所有参与者的能力。而使用 P2P 模式充分地利用了所有节点的带宽资源，使得并行下载能力得到了极大的扩展。

19.2 网络拓扑结构

P2P 技术将各个 peer 互相结合成一个网络，共享其中的带宽，共同处理其中的信息。

在 P2P 工作方式中，每一个客户终端既是客户机又是服务器。以共享下载文件为例，下载同一个文件的众多用户中的每一个用户终端只需要下载文件的一个片段，然后互相交换，最终每个用户都能得到完整的文件。

P2P 技术主要是利用多个终端在下载的时候复用资源，也就是说，在下载的时候每个终端同时也充当着服务器的角色。在协议基础上，P2P 技术把文件进行拆分成块，然后又分成片，最终以片为基本单位进行传输。P2P 最根本的思想，同时也是它与 C/S 最显著的区别，在于网络中的节点既可以获取其他节点的资源或服务，同时又是资源或服务的提供者，即兼具客户机和服务器的双重身份。一般 P2P 网络中每一个节点所拥有的权利和义务都是对等的，包括通信、服务和资源消费。

从技术上讲，P2P 技术一般都是基于成熟的 TCP/IP 协议的，并且借鉴服务器应用中许多成熟的技术。从层次上划分，P2P 应该属于网络应用层技术，与 Web 和 FTP 等应用是并列的。然而，P2P 技术又比这些应用更复杂。P2P 技术属于覆盖层网络（Overlay Network，ON）的范畴，是相对于 C/S 模式来说的一种网络信息交换方式。P2P 的核心思想是上网用户能在各个节点之间进行自由的、不受主服务器控制的、直接的信息交流，即“非中心化”思想。

P2P 网络结构可分为集中式、分布式和混合式三种，分布式又可分为非结构化和结构化两种。

19.2.1　集中式结构模式

在集中式结构模式（第一代 P2P 网络）中，设有一个中心服务器，其作用是负责记录共享信息和回答对这些信息的查询。每一个对等实体负责与中心服务器之间的信息共享和通信，并根据需要从其他对等实体上下载信息资源，其拓扑结构如图 19-1 所示。

集中式 P2P 模式将所有网上提供的资料都分别存放在提供该资料的客户机上，服务器上只保留索引信息，此外，服务器与对等实体、对等实体之间都具有交互能力。

集中式 P2P 模式的典型代表有 Napster 等，这种结构模式的主要优点是：部署和维护简单、查找效率高，并且可以进行模糊查询；由于资源的发现依赖中心化的目录系统，发现算法灵活、高效，并能够实现复杂查询。集中式 P2P 模式的主要缺点是：类似于传统 C/S 结构，容易造成单点故障，可靠性和安全性较低。

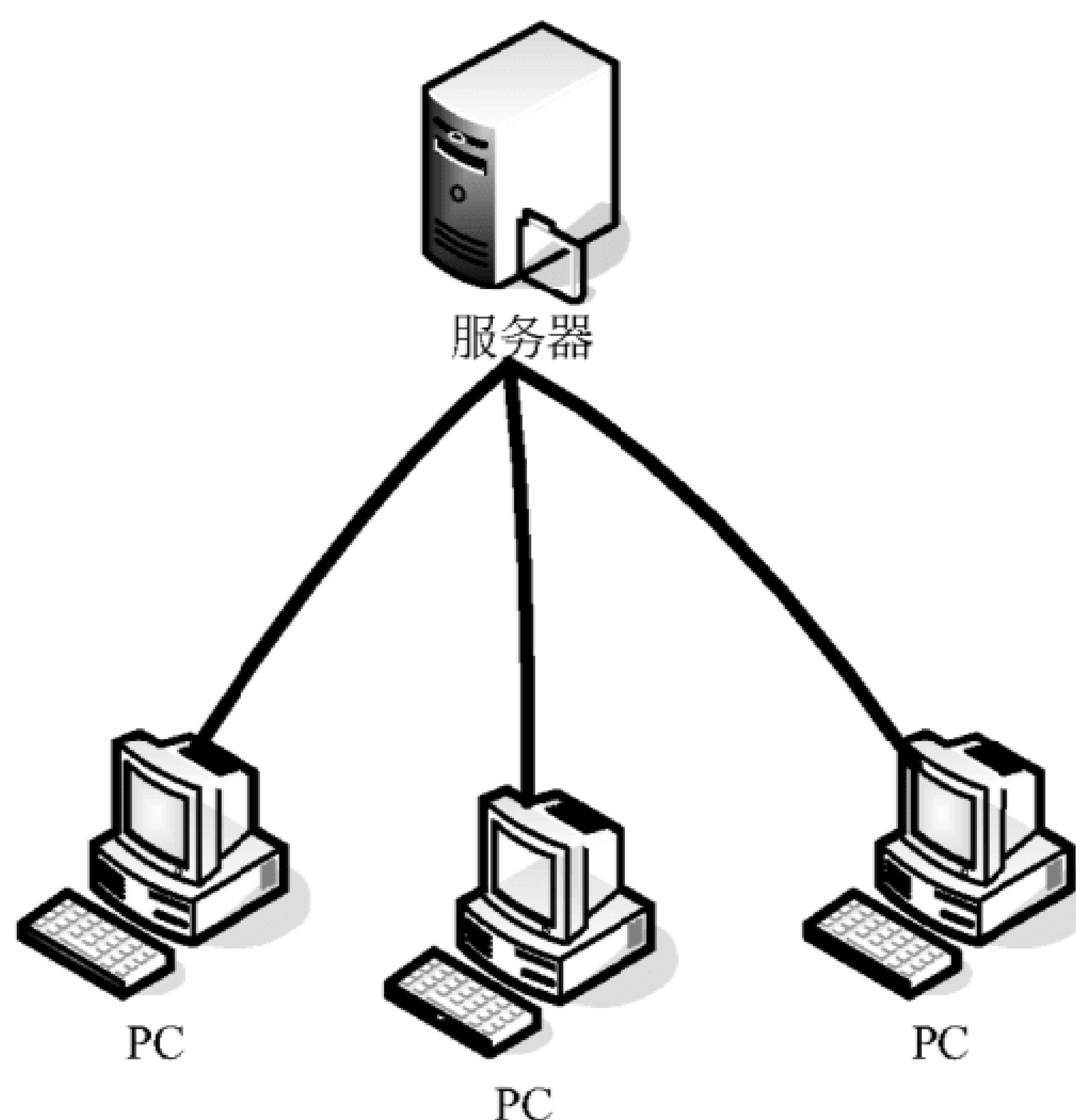


图 19-1　集中结构模式

19.2.2 分布式非结构化模式

分布式非结构化模式（第二代 P2P 网络）采用随机图的方式组织网络，从而能够较快发现目的节点，具有较好的可用性，容易维护，并支持复杂的查寻，但不能保证查询结果的完整性。为了保证查询结果的完整性，有些 P2P 网络需要维护一个中心目录，但这样就大大限制了网络的可扩展性，而且在很多情况下也不可行。

分布式 P2P 模型避开了集中式结构的典型缺点，但也存在很多弊端，主要表现在以下方面：

（1）搜索请求要经过整个网络或至少是一个很大的范围才能得到结果，正因为如此，这种模式占用很多带宽，而且需要花费很长时间才能有返回结果。

（2）随着网络规模的扩大，通过扩散方式定位对等点及查询信息的方法将造成网络流量急剧增加，从而导致网络拥塞。

在分布式非结构化对等网中，对等机通过与相邻对等机之间的连接遍历整个网络体系。每个对等机在功能上都是相似的，并没有专门的服务器，而对等机必须依靠它们所在的分布网络来查找文件和定位其他对等机。分布式非结构化的 P2P 网络拓扑结构如图 19-2 所示。

PC
PC
PC
PC
PC
My PC
PC
PC

图 19-2 分布式非结构化模式

分布式非结构化 P2P 模式的典型代表主要有 Gnutella、Freenet、Kazaa 等，这种模式的主要特点是：对象查询是分布式和逐跳的，采用泛洪式查询直到成功或失败（超时）。分布式非结构化的 P2P 网络的优点是：自组织的可管理性得到了增强，并且支持模糊查询；缺点是：因受到泛洪、回溯等方式的消息传递的资源定位模式的制约使得网络规模的可缩放性较差，查询效率低。

19.2.3 分布式结构化模式

集中式 P2P 结构化模式有利于网络资源的快速检索，且服务器可以扩展，但是其中心化的模式容易遭到直接的攻击；分布式非结构化的 P2P 模式解决了抗攻击问题，但是又缺乏快速搜索和可扩展性。

分布式结构化模式（第三代 P2P 网络）主要是采用分布式散列表技术来组织网络中的节点。这种模式吸取了中心化结构和分布式非结构化拓扑的优点，选择性能较高的节点作为超级节点，这些超级节点的在运算处理、存储、带宽等方面有较高的性能。在各个超级节点上存储了系统中其他部分节点的信息，发现算法仅在超级节点之间转发，超

级节点再将查询请求转发给适当的叶子节点，超级节点之间构成一个高速转发层，超级节点和所负责的普通节点构成若干层次。其拓扑结构如图 19-3 所示。

在分布式结构化 P2P 系统中，文件按照 P2P 拓扑中的逻辑地址精确地分布在网络中。这类系统的典型代表有 Tapestry、Pastry、Chord 和 CAN 等，以及基于这些系统的一些其他文件共享和检索方面的研究实验系统。在这类系统中，每个节点都具有虚拟的逻辑地址，并根据地址使所有节点构成一个相对稳定而精致的拓扑结构。在此拓扑上构造一个存储文件的 DHT，文件根据自身的索引存储到哈希表中。每次检索也是根据文件的索引在 DHT 中搜索相应的文件。生成文件索引的方法有三种：根据文件的信息生成的哈希值；根据文件包含的关键字生成关键字索引；根据文件的内容向量索引。

在分布式结构化 P2P 网络中，网络的节点拓扑关系有严格定义，节点之间通过一定的协议来维护叠加网络的拓扑结构。由于采用了确定性的拓扑结构，该种网络提供高效并具确定性的查询。只要目的节点存在于网络中，发现的准确性就会得到保证，但维持网络的拓扑结构将消耗较多的网络资源。

分布式结构化 P2P 模式的主要优点是：在资源管理过程中同时拥有自组织特性、规模的强可缩放特性以及部署的廉价性等等，为规模庞大的资源整合及共享提供了可能性；缺点是：节点仅存在局部视图，缺少权威第三方的控制，不支持模糊查询。

19.2.4　混合结构模式

混合结构模式（第四代 P2P 网络）引入了网络分层的思想，在 P2P 网络中采取多级分层结构，实现了多种网络结构并存的网络模型设计，提高了网络的可扩展性和透明性，并降低了主干网络通信流量。在此基础上提出的管理节点模式和关键值匹配方案进一步改善了 P2P 网络不可管理现状，并为该模型从理论到实用的转化奠定了基础。

在混合结构中，也是选择在处理、存储、带宽等方面性能较高的节点作为超级节点，方式与分布式结构化模式相同。混合结构模式的网拓扑如图 19-4 所示。

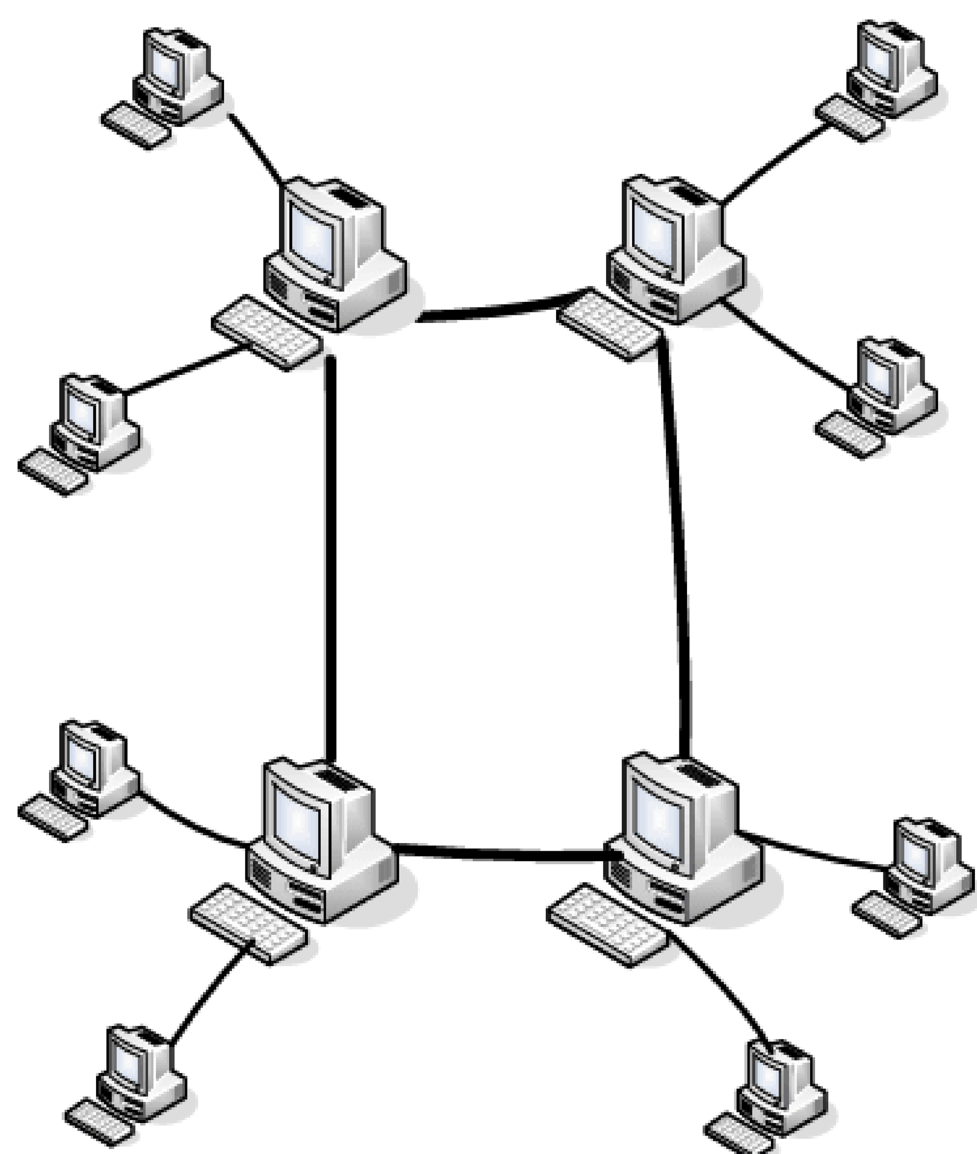


图 19-3　分布式结构化模式

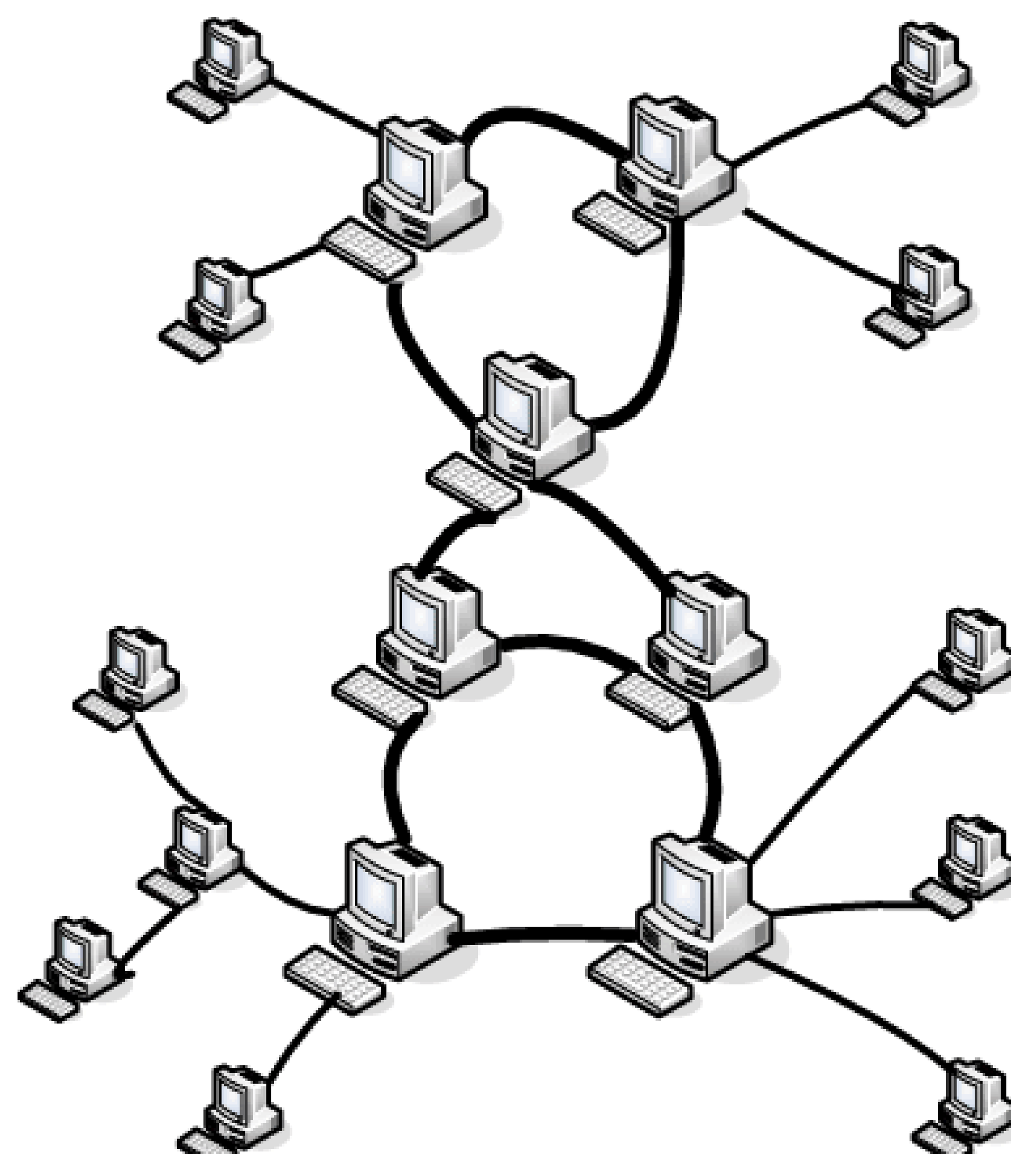


图 19-4　混合结构模式

混合结构模式的主要特点是多级分层结构，其典型代表有 Kazaa。这种模式的主要优点是：吸取了中心化结构和全分布式非结构化拓扑的优点；主要缺点是：由于超级节点本身的脆弱性也可能导致其簇内的节点处于孤立状态，因此，这种局部索引的方法仍然存在一定的局限性。

作为本节的结束，表 19-1 从几个方面对 P2P 网络的 4 种拓扑结构的性能进行了比较。

表 19-1　P2P 网络拓扑结构比较

比较内容	集中式	分布式非结构化	分布式结构化	混合式	备　　注
可靠性	差	好	好	一般	中央索引服务器容易造成单点故障
可扩展性	差	较差	好	一般	
可维护性	简单	简单	比较简单	一般	随着网络规模的扩大，对中央索引服务器进行维护和更新的费用将急剧增加
发现算法效率	灵活高效	一般	高	一般	
复杂查询	支持	支持	不支持	支持	
抗攻击性	差	好	好	一般	如果负载过量存储就容易遭到直接的攻击
版权纠纷问题	容易出现	不易出现	不易出现	不容易出现	中央索引服务器的存在常引起版权问题上的纠纷，服务提供商容易被追究法律责任
对大型网络应用的适合性	差	较差	好	好	

19.3　P2P 的关键技术

P2P 是一种基于互联网环境的新的应用型技术，包括下列主要内容：

（1）资源查找与定位技术。请参考 19.1.4 节的介绍。

（2）网络地址转换（Network Address Translators，NAT）路由转换技术。NAT 是在 IP 地址日益缺乏的情况下产生的，它的主要目的是为了地址重用。NAT 分为两大类，基本的 NAT 和网络地址/端口转换（Network Address/Port Translator，NAPT）。NAPT 不但会改变经过这个 NAT 设备的 IP 数据报的 IP 地址，还会改变 IP 数据报的 TCP/UDP 端口。

（3）IP 地址解析技术。提供在现有硬件逻辑和底层通信协议上的端到端定位（寻址）技术，建立稳定的连接。

（4）安全技术。P2P 网络应用比其他应用要更多的考虑那些低端 PC 的互联问题，加上 P2P 网络结构的特殊性，安全管理是个难点，需要进行全面的安全技术做后盾才能

保证 peer 的节点安全。

（5）应用层上的数据描述和交换技术，如 XML、SOAP、UDDI 等。

19.3.1　P2P 的技术特点

相对于传统的互联网技术来说，P2P 网络技术具有以下特点：

（1）非集中化。网络中的资源和服务分散在所有节点上，信息的传输和服务的实现都直接在节点之间进行，避免了可能的服务器瓶颈，强调用户端所有权和对数据资源的控制，每个 Peer 都是平等的参与者，既是服务器又是客户机，如何表现取决于用户的要求，网络应用由使用者自由驱动。

（2）可扩展性。由于 P2P 网络结构的分布性特点，节点的加入很容易，因此，网络结构和规模的扩充很容易，系统整体资源和服务能力也容易同步扩充，较容易满足用户的需要。

（3）健壮性。P2P 由于其完全分布式架构，网络中的节点既可以获取其他节点的资源或服务，同时又是资源或服务的提供者，不依赖于少数集中控制节点，具有比传统的 C/S 网络更好的健壮性和抗毁性，成为搭建高健壮性网络的有效方式。由于服务是分散在各个节点之间进行的，部分节点或网络遭到破坏对其他部分的影响很小。P2P 网络一般在部分节点失效时能够自动调整整体拓扑，保持其他节点的连通性。P2P 网络通常都是以自组织的方式建立起来的，允许节点自由地加入和离开。P2P 网络还能够根据网络带宽、节点数、负载等变化不断地做自适应式的调整。

（4）匿名与隐私保护。在 P2P 网络中，由于信息的传输分散在各节点之间进行而无需经过某个集中环节，用户的隐私信息被窃听和泄漏的可能性大大缩小。重要目的是让人们使用系统时不用关心法律问题和其他节外生枝的问题，而且可能使有关信息内容的审查制度形同虚设。

（5）自组织。P2P 系统的组织（约束/冗余）自然本能地增加，不通过环境，也不包含其他外部系统来增加控制。

（6）所有权成本。P2P 的前提是共享所有权，集中化计算机存储信息的削减也减少了所有权和维护成本。

（7）Ad-hoc 连接。Ad-hoc 连接是一种特殊的、多跳的、共享带宽的移动无线网络，Ad-hoc 结构是一种省去了无线接入点（Access Point，AP）而搭建起的对等网络结构，只要安装了无线网卡，计算机彼此之间即可实现无线互联。

（8）高性价比。性能优势是 P2P 被广泛关注的一个重要原因。采用 P2P 架构可以有效地利用互联网中散布的大量普通节点，将计算任务或存储资料分布到所有节点上，利用其中闲置的计算能力或存储空间达到高性能计算和海量存储的目的，通过利用网络中的大量空闲资源，可以用更低的成本提供更高的计算和存储能力。

（9）安全。Peers 和共享对象间具有信任链、会话密钥交换模式、加密和数字摘要与

签名等安全功能。

（10）透明性和可用性。P2P 透明性主要形式是位置透明性，在访问、并发、复制、失效、移动、扩展等方面也具有透明性。

（11）故障适应力。P2P 主要目标之一是避免中心节点失效，因此具有较强的故障适应能力。

（12）交互能力。P2P 根据 Sockets、HTTP 协议进行通信交互，具有很强的交互能力。

（13）负载均衡。P2P 网络环境下由于每个节点既是服务器又是客户机，减少了对传统 C/S 结构服务器计算能力、存储能力的要求。同时，因为资源分布在多个节点上，更好地实现了整个网络的负载均衡。

19.3.2 P2P 的流量特性

随着互联网应用范围的扩大和应用程度的深入，P2P 应用已经成为网络带宽的最大消费者，已成为运营商业务的主流，对底层网络造成了巨大的影响。

P2P 网络的发展首先对于个人用户接入网络的性能提出了更高的要求。P2P 的流量呈现出与传统流量不同的特性，具有分布非均衡的特性、上下行流量的对称特性、流量的隐蔽性、数据集中性等。具有高带宽的用户通常会以更长的时间为其他节点提供下载服务，上下行对称流量已经成为 P2P 网络流量区别于其他流量的主要特征。由于 P2P 流量特征具有上下行流量对称的特性，这使得直接面向用户的接入网络需要相应提高所能承载上行流量的能力。

P2P 流量还具备很强的隐蔽特性，它们通常使用随机端口或用户自定义端口，无法通过简单的端口识别 P2P 流量，目前常用的方法是通过特征码检测的方式识别 P2P 流量。P2P 相对随机的端口号，使得企业难以对内部的网络实行有效的监测和管理，加大了日常维护的难度。对于 Internet 服务提供商（Internet Service Provider，ISP），P2P 应用的影响不仅增加了网络升级的难度，同时也将降低了网络的总体性能以及 P2P 本身的服务质量。

用户可以选择高带宽接入以正常使用各种 P2P 应用，企业用户可以通过谨慎的企业内部安全规范的制订来保证网络的正常使用，而 ISP 成为 P2P 应用产生流量的最终承担者，设备升级速度加大，维护费用升高。有效识别和管理网络中的 P2P 流量成为 ISP 最为关心的问题。P2P 应用的特征码处在不断变化中，加深了 P2P 流量的这种隐蔽特性。

19.4 P2P 的应用

P2P 引导网络计算模式从集中式向分布式偏移，也就是说，网络应用的核心从中央服务器向网络边缘的终端设备扩散：服务器到服务器、服务器到 PC、PC 到 PC，PC 到无线应用协议（Wireless Application Protocol，WAP）手机等，所有网络节点上的设备都

可以建立 P2P 对话。这使人们在 Internet 上的共享行为被提到了一个更高的层次，使人们以更主动、深刻的方式参与到网络中去。

19.4.1　主要应用

从目前来看，P2P 技术的典型应用如下。

1．共享和交换

共享和交换是最引人注目的 P2P 应用。事实上，激起 P2P 革命的就是文件共享的应用，而文件交换的需求直接引发了 P2P 技术热潮。在传统的 Web 方式中，要实现文件交换需要服务器的大力参与，通过将文件上传到某个特定的网站，用户再到某个网站搜索需要的文件，然后下载，这种方式的不便之处不言而喻。电子邮件虽然方便了个人间文件传递问题，却无法解决大范围的交换。这也是 Web 的重要缺陷。

典型的文件传输和内容共享软件有 Napster、Gnutella、eDonkey、emule、BitTorrent 等。

2．内容搜索技术

P2P 文件共享首先要解决文件定位的问题，但是，基于 P2P 的文件搜索技术可以独立出来，成为传统的搜索引擎等系统强大的搜索工具。可利用 P2P 技术开发互联网上高级智能搜索引擎技术。一个好的搜索引擎不再仅凭借数据库大小、更新频率、检索速度、对多语言的支持这几个基本特性来衡量，随着数据库容量的不断膨胀，如何从庞大的资料库中精确地找到正确的资料，被公认为是下一代搜索技术的竞争要点。智能搜索可以通过对搜索内容相关性的自动学习，来提高搜索结果的准确度。

另一个颇受瞩目的搜索技术就是将 P2P 技术应用到网页的检索中。通过共享所有硬盘上的文件、目录乃至整个硬盘，用户搜索时无需通过 Web 服务器，不受信息文档格式的限制，即可达到传统目录式搜索引擎无可比拟的深度（传统引擎只能达到 20%～30% 的网络资源）。随着 P2P 技术的深入研究和应用，最终的搜索引擎将是跨平台、多格式的、高度智能化的搜索工具。

典型的数据搜索软件有 Infrasearch、Pointera、KuroM3Kuro 等。

3．对等计算

对等计算是分布式计算的思想在广域网上的延伸，目的是将网络上的 CPU 资源共享，把网络中众多的普通计算机中暂时不用的计算能力累计起来，用以执行以往需要超级计算机来完成的任务。采用 P2P 技术的对等计算，正是把网络中的众多计算机暂时不用的计算能力连接起来，使用积累的能力执行超级计算机的任务。任何需要大量数据处理的行业都可从对等计算中获利，如天气预报、动画制作、基因组的研究等。有了对等计算之后，就不再需要昂贵的超级计算机了。从本质上而言，对等计算就是网络上 CPU 资源的共享。

可以将 P2P 看作一个松耦合的分布式计算系统，可以集中控制，也可以是纯 P2P 架

构。在对等计算中，大型的计算任务被分解成很多个小的分片，分别分配给网络中的节点独立执行，非常适合那些可以分解的计算密集性任务。例如，2002 年 9 月破解了 RSA 公司悬赏的 RC5-64 密码的组织，正是利用对等计算技术，集合了互联网上的 331 252 台计算机，才完成了这一巨大的计算量。

4．协同工作与在线交流

P2P 技术的出现，使得互联网上任意两台 PC 都可建立实时的联系，建立了一个安全、共享的虚拟空间，人们可以进行各种各样的活动，这些活动可以是同时进行，也可以交互进行。协同工作依托在网络上，但以传统的 Web 方式实现，往往给服务器带来极大的负担，并造成了昂贵的成本支出。而采用 P2P 技术，可以在互联网上任意两个用户之间建立实时的联系和信息传输，避免了中央服务器产生的网络和处理延迟及性能瓶颈，因而能够更方便、高效地实现用户之间的协同。

即时通（Instant Messaging，IM）工具正是实现了用户之间的直接交流，受到了互联网用户的极大欢迎，可以说已经是无处不在。目前很多公司正努力将这种方式应用到企业级的协同工作平台中来，已经推出了一些产品。由于其具有成本低廉、平均事务处理能力较高、可动态扩展等特点，并能够有效地提高信息交流和沟通效率，未来 P2P 技术在企业级协同工作领域将有着很好的应用前景。

企业可以利用 P2P 技术进行协同办公，包括两大方面，一是企业内部的协同，包括员工与员工、部门与部门、员工与部门之间的协同，无论部门及员工处在何种地理位置，只要拥有网络，双方存在信息沟通的要求，就可以利用 P2P 软件协调双方的行为。信息的种类及行为目的可以是多种多样的，例如，日程安排、通知发布、单据的审批、文件传阅和分发、方案的评比（表决）、计划协调等，这些可以通过电子文件表达的信息均可以通过软件来表达，并以此将使用者联系在一起；二是企业与企业之间的行为协同，这种协同的多样性和复杂性要高于企业内部的协同。从简单的会议日程安排、公文往来，到报价、询价订货系统、订单跟踪、电子化交易等。

企业内部的员工与企业外部的伙伴或客户也可利用 P2P 协同体系建立企业门户，通过企业门户这一平台找到相关的人（工作人员），建立各类信息交流通道，实现不同的协作目标。

典型的协同工作类软件有 Netbatch、Groove 等，典型的即时通信产品有 ICQ/OICQ、AOL Instant Messenger、Yahoo Pager、MSN Messenger 等。

5．网络存储

P2P 带来的一个变化就是内容正在从“中心”走向“边缘”，也就是说，内容将不是存在几个主要的服务器上，而是存在所有用户的 PC 上。这就为网络存储提供了可能性，可以将网络中的剩余存储空间利用起来，实现网络存储。

人们对存储容量的需求是无止境的，提高存储能力的方法通常有两种，一是更换能力更强的存储器，二是把多个存储器用某种方式连接在一起，实现网络并行存储。网络

存储比较容易实现，而且价格相对便宜。相对于现有的网络存储系统而言，应用 P2P 技术将会有更大的优势。因为 P2P 技术的主体就是网络中的客户机，数量众多，这些客户机的空闲存储空间是很多的，把这些空间利用起来，让数据分别存储在各个节点上以实现网络存储。

6．P2P 群集与 VPN

P2P 应用可以扩展到多点的群集，形成互联网中一个虚拟的子网，构成一个精简的虚拟专用网（Virtual Private Network，VPN），通过对 P2P 用户端软件的操作，用户就可以主动地选择并加入不同的 VPN 中。这种方式可以为行业化的目录服务、信息服务与电子商务所利用。

7．远程监控和调试

P2P 技术为远程监控和调试的应用开辟了新的天地，利用这项功能，可以通过互联网或手机遥控操作家用空调、计算机，甚至锅炉等。

8．宽带网及无线移动网应用

利用 P2P 技术结合宽带网技术可以开展许多音频、视频和无线网方面的应用，如电话会议、视频会议、远程教育与培训等。

9．流媒体直播与点播

P2P 技术还可以实现流媒体直播，P2P 流媒体直播软件主要有 Coolstreaming、AnySee、Gridmedia、PPLive 等。

流媒体点播与流媒体直播不同，在流媒体直播中，用户在观看直播节目时不能选择观看指定片段；而在流媒体点播中，用户还可以实现节目指定功能。目前，实现 P2P 流媒体点播的软件有 GridCast、PPStream 等。

希赛教育专家提示：广播影视资料内容的分发主要采用两种方法：一种方法是先下载，下载后再观看，这种方法被称为播客（Podcast）；另一种方法就是用流媒体的方式边下载边收看。P2P 技术对这两种方式都支持。

10．IP 层语音通信

相对于传统的公共交换电话网络（Public Switched Telephone Network，PSTN）业务来说，IP 层语音通信（VoIP）是一种全新的网络电话通信业务，具有扩展性好、部署方便、价格低廉等明显的优点。采取 P2P 技术，根据通信双方网络进行动态自适应的链路控制与消息转发是可行的解决方案。这方面应用的代表产品有 Skype，它是一款典型的 P2PVoIP 软件。

11．网络游戏平台

基于 P2P 方式的网络游戏是一种很有前景的应用。大型网络在线游戏和网络对战游戏是不少“网虫”的至爱。但由于服务器能力有限，大型网络在线游戏往往需要限制场景人数或不断增加服务器，而网络对战游戏也必须局限在局域网内进行或依赖独立的服务器端程序及机器实现 Internet 上的电子竞技。目前，已有研究人员将 P2P 技术引入网

络游戏和网络游戏支撑平台中。较为成功的 P2P 游戏平台有 PKTown 等。

19.4.2 流行的 P2P 软件

目前，比较流行的 P2P 软件有很多，本节简单介绍 BT、eMule、迅雷、PPLive 和 Skype。

1. BT

比特流（BitTorrent，BT）是一种基于 P2P 技术将文件在大量互联网用户之间进行共享与传输的无结构的网络协议，对应的客户端软件有 BitTorrent、BitComet 和 BitSpirit 等。由于其实现简单、使用方便，得到了广泛使用。

BT 发布需要做种，且必须上传服务器，然后发表种子文件，用户必须通过种子文件链接。BitTorrent 中的节点在共享一个文件时，首先将文件分片并将文件和分片信息保存在一个流（Torrent）类型文件中，这种节点被形象地称作“种子”节点。其他用户在下载该文件时根据 Torrent 文件的信息，将文件的部分分片下载下来，然后在其他下载该文件的节点之间共享自己已经下载的分片，互通有无，从而实现文件的快速分发。由于每个节点在下载文件的同时，也在为其他用户上传该文件的分片，所以整体来看，不会随着用户数的增加而降低下载速度，反而下载的人越多，速度越快。

2. eMule

eMul（电骡）是一个叫 Merkur 的人基于当时的 eDonkey2000 客户端功能于 2002 年 5 月 13 日开发出来并进行命名的 P2P 软件，它是基于 eDonkey 协议改进后的协议，同时兼容 eDonkey 协议，至今已成为世界上非常出色的点对点文档共享与传输的客户端软件。

eMule 的排队机制和上传积分系统有助于激励人们共享并上传给他人资源，以使自己更容易、更快速地下载自己想要的资源。

eMule 提供了一个大范围的搜索方式，包含了各地的服务器搜索、基于 Web 的搜索等。eMule 的搜索非常灵活，并有使用信息及好友系统，能传送信息到其他的客户端并可将他们加为好友，当有好友上线时，就能在好友列表中看到。

每个 eMule 客户端都预先设置好了一个服务器列表和一个本地共享文件列表，客户端通过 TCP 连接到 eMule 服务器进行登录，得到想要的文件信息以及可用的客户端信息。一个客户端可以从多个其他的 eMule 客户端下载同一个文件，并从不同的客户端取得不同的数据片段。

eMule 与 BT 的比较如下。

（1）共同点：eMule 和 BT 都是标准的 P2P 软件，基本原理类似，客户端通过索引服务器获得文件下载信息。

（2）不同点：eMule 同时允许客户端之间传递服务器信息，BitTorrent 只能通过索引服务器或 DHT 获得。eMule 共享的是整个文件目录，而 BitTorrent 只共享下载任务，这

使得 BitTorrent 更适合分发热门文件，eMule 倾向于一般热门文件的下载。

3．迅雷

迅雷结合了传统的 HTTP 多线程下载和镜像搜索技术。迅雷的技术主要分成两个部分，一部分是对现有 Internet 下载资源的搜索和整合，将现有 Internet 上的下载资源进行校验，将相同校验值的 URL 信息进行聚合，当用户点击某个下载连接时，迅雷服务器按照一定的策略返回该 URL 信息所在聚合的子集，并将该用户的信息返回给迅雷服务器；另一部分是迅雷客户端通过多资源、多线程下载所需要的文件，提高下载速率。迅雷高速稳定下载的根本原因在于同时整合多个稳定服务器的资源实现多资源、多线程的数据传输。多资源、多线程技术使得迅雷在不降低用户体验的前提下，对服务器资源进行均衡，有效降低了服务器负载。

每个用户在网上下载的文件都会在迅雷的服务器中进行数据记录，如果有其他用户在下载同样的文件，迅雷的服务器会在它的数据库中搜索曾经下载过这些文件的用户，服务器再连接这些用户，通过用户已下载文件中的记录进行判断，如果用户下载文件中仍存在此文件（如果文件被重新命名或改变保存位置，则无效），用户将在不知不觉中扮演下载中间服务角色，并上传文件。

4．PPLive

PPLive 软件是视频播放工具，它的工作机制和 BitTorrent 十分类似，PPLive 将视频文件分成大小相等的片段，第三方提供播放的视频源，用户启动 PPLive 以后，从 PPLive 服务器获得频道的列表，用户点击感兴趣的频道，然后从其他节点获得数据文件，使用流媒体实时传输协议（Real-time Transport Protocol，RTP）和实时传输控制协议（Real-time Transport Control Protocol，RTCP）进行数据的传输和控制。将数据下载到本地主机后，开放本地端口作为视频服务器，PPLive 的客户端播放器连接此端口，任何同一个网段的用户都可以通过连接这个地址收看到点播的节目。

5．Skype

Skype 是在 Kazaa 的基础上开发的网络语音沟通工具，它可以提供免费的、高清晰的语音对话，也可以用来拨打国内、国际长途，还具备即时通信所需的其他功能，如文件传输、文字聊天等。

Skype 有两种类型的节点：普通节点和超级节点。普通节点是能传输语音和消息的一个功能实体，超级节点则类似于普通节点的网络网关，所有的普通节点必须与超级节点连接，并向 Skype 的登录服务器注册以加入 Skype 网络。Skype 的登录服务器上保存了用户名和密码，并且可授权特定的用户加入 Skype 网络。

Skype 能够穿越地址转换设备，并有极强的防火墙穿透能力。Skype 能够在最小传输带宽（32kbps）的网络上提供高质量的语音，具有超清晰语音质量、高保密性、免费多方通话等优点。

19.5 存在的问题与解决办法

P2P 技术与传统的网络技术相比最大的优势如下。

（1）将繁重的计算分配到各个节点上，利用每个节点宽裕的计算能力和存储空间，聚合实现强大的服务，每个用户都可以直接和其他用户进行连接互访，从而突破了服务器和带宽的瓶颈，使用的用户越多，网络互联的效果则越好。

（2）能够提供可靠的信息查询。

（3）加强和改进了许多原有的应用，使资源得到了充分利用和最大化的共享。

（4）系统可扩展性强，且能避免传统的服务器连接带宽的限制。

（5）不存在中心节点失效的问题，网络更加健壮，当一部分节点连接失败之后，其余的节点仍然能形成完整的网络。

（6）不需要中心服务器的维护成本，各个节点能自主管理本地资源。

（7）通过聚合众多节点的资源，提供超级服务器的强大计算能力，实现超级计算功能。

但是，P2P 技术的发展道路很不平坦，有很多问题亟待解决，如标准不统一问题、共享与版权问题、安全与管理问题、垃圾信息与网络带宽问题等。

1．标准问题

在国际上，2000 年 8 月成立了 P2P 工作组，成员包括 Intel、IBM、HP 等大企业，目标集中在 P2P 技术的标准、安全性及可靠性等。但到目前为止，P2P 技术仍然缺乏一个统一的标准与规范，很难实现 P2P 应用之间的统一资源定位和统一路由，因此，很难提升 P2P 的整体性能。在国内，中国通信标准化协会已开始进行 P2P 相关标准的制订，并针对 P2P 和科普感知等问题展开了相关研究。

2．共享与版权问题

版权问题一直是 P2P 发展的一个突出的问题，就像 Napster 的出现冲击着唱片公司的利益一样，大多数 P2P 服务都将不可避免地和知识产权发生冲突，每一个提供文件共享服务的 P2P 公司都不得不认真审视这个问题。

如今，知识产权保护问题在 P2P 共享网络中普遍存在着，尽管目前 Gnutella、Kazaa 等 P2P 共享软件宣传其骨干服务器上并没有存储任何涉及产权保护的内容的备份，而仅仅是保存了各个内容在互联网上的存储索引，但是，P2P 共享软件的繁荣加速了盗版媒体的分发，提高了知识产权保护的难点。

3．垃圾信息问题

由于 P2P 网络的用户众多，当某个用户进行搜索时，自然会得到大量的搜索结果，而除了少数有用的信息以外，其他大多数的信息可能都属于垃圾信息。在缺乏统一的管理的情况下，P2P 网络很难对搜索结果进行排序，用户将不可避免地陷入垃圾信息的汪

洋大海。现在，已经有公司尝试着将人工智能技术、专家数据库技术引入 P2P 网络中，希望能够克服垃圾信息的困扰。

4．网络流量、带宽与负载均衡问题

P2P 网络中的节点本身往往是计算能力相差较大的异构节点，每一个节点都被赋予了相同的职责而没有考虑其计算能力和网络带宽，局部性能较差的节点将会导致整体网络性能的恶化，在这种异构节点的环境中容易造成大负荷的网络流量，占用网络带宽，难以实现优化的资源管理和负载平衡，尤其是在传送大量的音频和视频文件的时候。

同时，由于用户加入或离开 P2P 网络的随意性，使得用户获得目标文件具有不确定性，导致许多并非必要的文件下载，而造成大量带宽资源的滥用。如何有效管理 P2P 应用所带来的巨大流量、如何进行负载均衡是网络运营商面对的重大课题。可通过缓存或流量工程的方法来减少 P2P 流量对网络本身的影响，同时利用 P2P 技术优化网络性能，例如，将 P2P 覆盖网络路由的优点引入到传统的网络路由中，可以实现更好的负载均衡。

5．安全性、可靠性与管理问题

安全问题对于 P2P 技术的发展至关重要，可以说关系到 P2P 的成败。P2P 网络采用的分布式结构在提供扩展性和灵活性的同时，也使它面临着巨大的安全挑战：它需要在没有中心节点的情况下提供身份验证、授权、安全传输、数字签名、加密等机制。如何在 P2P 网络中实现数据存取安全、路由安全、用户身份认证和身份管理都需要进一步研究。

有关这方面的内容，将在 19.6 节详细讨论。

19.6　P2P 与网络安全

在 19.5 节中，介绍了 P2P 技术目前所存在的问题以及解决办法，其中，网络安全与安全管理问题尤为严峻。

1．信任与激励机制问题

P2P 网络和人际网络具有一定的相似性。每个 P2P 网络都是众多参与者按照共同兴趣组建起来的一个虚拟组织，节点之间存在着一种假定的相互信任关系，但随着 P2P 网络规模的扩大，这些 P2P 节点本身所特有的平等自由的动态特性往往与网络服务所需要的信任协作模型之间产生矛盾。在纯分布式对等网络中，没有中心控制，具有高度的动态性、自治性和异构性，即每个用户参与网络是随机的、自愿的，并且不同的用户有不同的能力和可靠性，由此导致不可靠的服务质量及大量欺诈行为，网络的可用性差。同时，激励作用的缺失使节点间更多表现出“贪婪”、“抱怨”和“欺诈”的自私行为，因此，P2P 中预先假设的信任机制实际上非常脆弱，这种信任也难以在节点之间进行推理，导致了全局性信任的缺乏，这直接影响了整个网络的稳定性与可用性。

目前，解决信任与激励机制问题的有效方法是在 P2P 网络中建立对等诚信机制，即

用户通过自己过去的经历或他人的推荐来选择符合自己要求的交互端的一种机制。对等诚信机制由于具有灵活性、针对性，并且不需要复杂的集中管理，可能是未来各种网络加强信任管理的必然选择，而不仅仅局限于对等网络。对等诚信机制的安全和有效性是影响网络可用性的关键因素，此机制能激励用户提供可靠高质量的服务，节省时间和通信开销，减少交互风险和损失，促进网络的良性发展。

2．病毒、木马与恶意软件

当使用 P2P 时，很难验证共享文件的来源，P2P 应用因此常被攻击者选择作为传递恶意代码的载体，导致 P2P 应用可能包含病毒、木马与恶意软件。由于在 P2P 网络中，逻辑相邻的节点地理位置可能相隔很远，而参与 P2P 网络的节点数量又非常大，每个节点防御病毒的能力是不同的，因此，通过 P2P 系统传播的病毒，涉及范围大，覆盖面广，只要有一个节点感染病毒，就可以通过内部共享和通信机制将病毒扩散到附近的邻居节点，在短时间内可以造成网络拥塞甚至瘫痪，通过网络病毒可以完全控制整个网络，从而造成很大的损失。

3．信息保护与匿名通信技术

由于 P2P 网络的特殊性，个人文档、账户等信息泄密的可能性很大，因此需要考虑如何保护用户的安全策略。在 Windows 环境下做好网络安全防护措施，可考虑 Windows 本地安全策略设置、共享权限设置、建立文件许可规则、加密离线文件、进行匿名隐私保护等。

目前，Internet 网络协议不支持隐藏通信端地址的功能。能够访问路由节点的攻击者可以监控用户的流量特征，获得 IP 地址，使用一些跟踪软件甚至可以直接从 IP 地址追踪到个人用户。安全套接字层（Secure Socket Layer，SSL）之类的加密机制能够防止其他人获得通信的内容，但是，这些机制并不能隐藏发送者的信息。

利用 P2P 无中心的特性可以为隐私保护和匿名通信提供新的技术手段。P2P 技术为解决 Internet 隐私问题开辟了一条新的可行方案。P2P 系统要求每个匿名用户同时也是服务器，为其他用户提供匿名服务。这意味着经过一个节点的消息可能是源于该节点，也可能是源于其他节点，很难决定是哪一种情况，攻击者不易找到明确的攻击目标，在一个大规模的环境中，任何一次通信都可能包含许多潜在的用户。另外，匿名通信技术如果被滥用将导致很多互联网犯罪而无法追究到匿名用户的责任。所以，提供强匿名性和隐私保护的 P2P 网络必须以不违反法律为前提，而在匿名与隐私保护和完善的法律监控之间寻找平衡。

4．安全管理存在较大的困难

P2P 网络由于缺少中心节点，节点的加入和退出给网络带来了不稳定性，所以整个网络的管理相对复杂。P2P 网络采用“乌托邦”式的管理方式，这种方式给了用户更多的自由，但是，这也陷入了“无政府主义”的困境。因此，给安全管理带来很大的困难，黑客、病毒、色情等内容容易非法进入网络。

另外，在实际应用方面，P2P 的运算结构很容易引发错误和故障。例如，在 P2P 网络中，用户可能会突然关闭其他人正在访问的计算机设备。同时，当计算机断开网络时就会出现其他人无法访问的情况，这些问题是安全管理的难点。

要解决 P2P 的安全问题，最根本的安全措施是采用授权、认证方法，建立比较完善的加密机制，严格进行身份检验和授权。同时，需要切实采用安全控制措施，以达到自身安全的目的。安全控制可采取动态随机端口控制措施结合防火墙过滤规则和加密手段进行。很多 P2P 应用需要在防火墙上打开特定的端口来允许接受共享文件，P2P 技术支持防火墙穿越，通过 HTTP 端口来承载 P2P 报文，因此，需要考虑防火墙具备深度报文检测（Deep Packet Inspection，DPI）能力，通过 DPI 扫描分类出流的应用层协议，标识出具体的 P2P 业务类型，并利用三层 Shaping 技术实施流量控制。早期的 P2P 应用都是固定的端口号，容易检测便于管理，后来逐渐发展到动态随机端口号，一些传统的检测方法失去了作用。目前新型 P2P 应用越来越具有反侦察的意识，采用一些加密的手法、伪装 HTTP 协议、传输分块等来逃避识别和检测。

尽管 P2P 存在许多缺陷，但 P2P 技术的飞速发展是势不可挡的。未来的网络将呈现大规模分布式、全球性计算和全球性存储的特征，从长远的趋势来看，对访问和传输服务的需求必将远远大于对计算功能的需要。P2P 技术是最有吸引力的个人通信技术，尤其是 P2P 与网格技术的结合，将是分布式计算技术最有潜力的发展趋势。

本章参考文献

[1] 佚名. P2P 技术档案. http://www.educity.cn/data/View_46848.html

[2] 金海，廖小飞. P2P 技术原理及应用. http://www.cww.net.cn/tech/html/2007/12/27/200712271319573500.htm

[3] Jack zhai. P2P 网络的结构学习. http://www.venustech.com.cn/NewsInfo/113/974.Html

第 20 章　网格计算与普适计算

在介绍网格计算和普适计算之前，先看一个故事。

某发射中心正在运用软件来模拟宇宙飞船“月亮 N 号”发射的过程，以预测什么情况下发射对宇航员最为有利。计算机研究与应用中心的程序员和科学家通过称为“飞天”的程序，模拟发射过程中对数百名“宇航员”各种器官的反映，而“宇航员”又是在不同天气、太阳强度、发射经纬度等数万种组合情况下被送入太空，每次实验都包括近 1000 万次模拟。即使是最快的计算机也难以应付这种复杂度的软件。“在一台服务器上运行那些模拟程序要花费若干年的时间”，该项目的软件总工程师如是说。

与此同时，国内有近 800 万台 PC 在大部分时间里是闲置的。尽管有些计算机的主人努力使用办公软件、观看电影或玩游戏，但他们机器的 CPU 的利用率仅仅为 10%或更低。一方面需要大规模的计算能力，另一方面却又是资源的巨大浪费。

科学家们选用了“模拟人体 3000”（一家软件开发商）的方案，建立了一套计算网格的入口。软件工程师们现在可以利用湖南省和广东省内的 20 万台计算机的计算能力，在一个星期内就完成了 1000 万次模拟计算。

这是梦吗？不，这就是网格计算。

20.1　网格计算概述

网格是把一个局域网、城域网甚至整个 Internet 整合成一台巨大的超级计算机，实现知识资源、存储资源、数据资源、计算资源、信息资源、专家资源的全面共享。虽然，网格可以分为各种地区性的网格（如酒泉发射中心网格、公司集团内部网格、局域网网格，甚至家庭网格和个人网格等），但事实上，网格的根本特征是资源共享而不是它的规模。

20.1.1　网格计算的定义

一般而言，网格按功能可以划分为以下三种类型：

（1）计算网格。着重于专门留出用于计算能力的资源。在这类网格中，大多数机器都是高性能服务器。

（2）拾遗网格。常用于大量桌面系统。机器上可用的 CPU 周期和其他资源被收集起来。通常授予桌面主人控制其资源的同时可以加入网格的权利。

（3）数据网格。负责容纳和提供跨多个组织的数据的访问。用户只要有权访问数据，

就不必关心数据位于哪里。

1998 年，FosLer Ian 和 Carl Kesselman 首次对网格进行定义：计算网格是一种由硬件和软件构成的信息技术基础设施，它能提供可靠的、可协调的、可扩展的和廉价的高端计算能力的访问。

2002 年，Foster Ian 从三个方面更清晰地定义了网格，他认为网格是一个满足如下三个条件的系统。

（1）网格能协调非集中式控制的资源，能集成和协调资源与用户在不同控制域内的活动。不同的控制域有使用集中计算的用户桌面，同一公司的不同的管理部门，或不同的公司等；同时，网格能解决包括安全、策略、认证、支付、成员资格等各种问题。

（2）使用标准的、开放的、通用的协议和接口，网格是由多用途协议和接口来构建的，该协议将能解决诸如鉴别、授权、资源发现和资源访问等一些基本问题。此处强调这些协议和接口的标准化和开放性是很重要的。

（3）提供非常高的服务质量，网格允许按协作的方式来使用其成分资源，以提供各种各样的服务内容，例如，反应时间、容许能力、可利用性和安全性，以及协作配置多重资源类型以满足复杂的用户要求等，这种组合系统的效用大大高于该系统的部分总和。

通俗地说，可以将网格计算比喻成电力网格。当给电器或其他用电设备接入电源时，可以使用正确电压的电力，但并不知道该电力的实际来源。当地的公用事业公司提供了接入由发电机和电源组成的复杂网络的接口，并且（在大多数情况下）为满足用户的能源需求提供了满意的服务质量。不需要每家每户或邻近地区使用和维护他们自己的发电机，电力网格基础设施提供了虚拟的发电机。该发电机高度可靠，并根据客户的要求来适应客户的电力需求。同样，在不同地方的存储、数据、空闲 CPU 等通过网格技术集成为一个整体。人们利用这些资源就像利用电源一样，不需考虑其来源和负荷情况。

20.1.2　网格系统的特点

网格和联网提供若干有限种类服务的服务器有天壤之别，一般来说，网格有如下特有的功能。

（1）异构性：网格包含多种异构的资源，这些资源可能跨越不同的地理位置和管理区域。多种异构的资源包括各种不同操作系统的主机、服务器或存储设备、数据库等。

（2）结构的不可预测性：与一般局域网系统和单机的结构不同，网格计算系统由于地域分布和系统的复杂性，使得整体结构经常变化，网格计算系统必须做到能够适应这种经常变化的结构。

（3）可适应性：传统的高性能计算机系统中，计算资源是独占的，而网格计算系统中的资源是异构的、分布的，而且经常发生变化，甚至发生故障，面对这些情况，网格计算系统应能做到动态的可适应性。

（4）可扩展性：网格计算系统的初期规模可能很小，随着参与网格计算的计算机系

统不断加入，系统的规模会越来越大。网格计算系统必须能够做到适应规模的增加、克服规模的膨胀而造成性能下降或计算的延迟。

（5）多级管理域：由于构成网格计算系统的超级计算资源通常属于不同的机构或组织，使用不同的安全机制，因此，需要不同的机构或组织共同参与解决多级管理域问题。

20.1.3 网格计算的应用领域

没有应用的技术是没有生命力的，正因为有了广泛的应用前景，网格得到了全世界科研人员和厂商的追捧。各行各业正在期待着网格技术的不断完善。总的来说，网格可以应用在如下几个方面。

（1）协同环境：可以连接多个虚拟环境，使不同位置的用户能进行交互、仿真。

（2）智能设备：可以连接大量的、分布的、远程的设备，进行实时处理和远程操作等。

（3）分布式并行计算：可以使多个异构计算机协同解决单机难以完成的任务，使不同性质的任务调度到最合适的计算机中去运行。

（4）桌面超级计算：可以将普通桌面用户和超级计算中心、大型数据库连接起来，用户可以不受距离限制地使用这些计算能力。

（5）军事仿真：可以模拟战场，通过详细的数字分析，掌握各战斗实体的状况和随之而来的战斗结果。

20.2 网络体系结构

网格主要由 6 个部分组成，分别是网格节点、数据库、贵重网络设备、可视化设备、宽带主干网和网格软件。网格的体系结构比 Internet 更能有效地利用网上的所有资源。如何让用户尽快得到所需信息而不管信息到底存放在什么地方，如何自动地从距离用户最近的服务器调入用户最需要的信息，如何存储和备份，如何协同工作网格上的多台服务器、PC 等，如何做到资源、计算、服务的平衡负载，这些都是设计网格系统的体系结构时所要考虑的问题。

网格体系结构必须要能够标识出网格组成的拓扑图，要能够清楚地说明网格整体是由哪些关键部分结合在一起形成的，还必须能够对各个部分的功能、目的、特点等进行清晰的描述，使人们能够了解各个组成部分的作用。显然，网格体系结构是网格的骨架和灵魂，是最核心的技术，只有建立合理的网络体系结构，才能设计出高效率的网格。

目前，主要的网格体系结构模型有五层沙漏模型、开放网格体系结构、计算机池模型等，其中五层沙漏是经典的模型。

1．五层沙漏模型

Globus 的五层沙漏模型把网格就是系统分为 5 层，分别是构造层、连接层、资源层、

汇集层和应用层。上层协议可调用下层协议的服务。网格内的全局应用都通过协议提供的服务来调用操作系统。网格内的全局应用都通过协议提供的服务来调用操作系统，如图 20-1 所示。

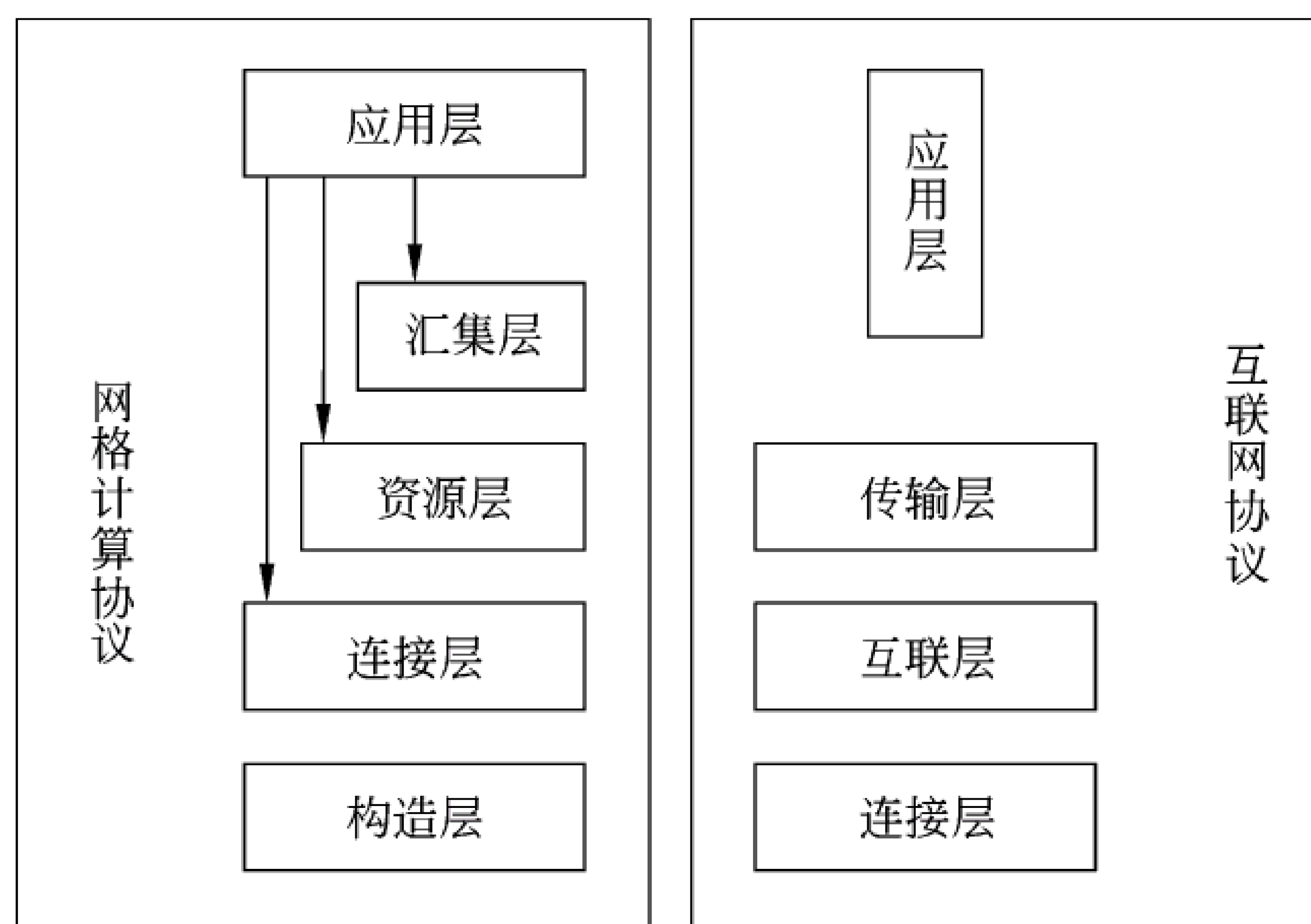


图 20-1　五层沙漏模型与 TCP/IP 的对照图

（1）构造层（fabric）：向上提供网格中可供共享的资源，它们是物理或逻辑实体。常用的资源包括工具包、计算资源、目录资源、存储资源、网格资源、分布式文件系统、分布式计算机池、计算机集群等。通常这些资源被封装成为有利于上层调用的形式。

（2）连接层（connectivity）：网格中的网络事务处理通信与授权控制的核心协议。构造层提交的各种资源间的数据交换都在连接层的控制下实现。各种资源之间的授权验证、安全控制也在连接层实现。

（3）资源层（resource）：对单个资源实施控制，与可用资源进行安全握手，对资源做初始化，监测资源运行状况，统计与付费有关的资源使用数据。本层相关的协议有信息协议（资源的结构和状态信息）和管理协议（磋商对资源的访问，分配、预留和监视、控制）。

（4）汇集层（collective）：将资源层提交的受控资源汇集在一起，供虚拟组织的应用程序共享、调用。为了对来自应用的共享进行管理和控制，汇集层提供目录服务、资源分配、日程安排、资源代理、资源监测诊断、网格启动、负荷控制、账户管理等多种功能。

（5）应用层（applications）：网格上用户的应用程序。应用程序通过各层的 API 调用相应的服务，再通过服务调用网格上的资源来完成任务。应用程序的开发涉及大量库函数，为便于网格应用程序的开发，需要构建支持网格计算的库函数。

2．开放网格体系结构

开放网格体系结构（Open Grid Services Architecture，OGSA）被称为下一代的网络

体系结构，在五层沙漏模型的基础上，结合 Web Service 技术提出来的。它是网格技术和 Web Service 的组合体。OGSA 最突出的思想是，将一切对象（包括数据库、服务器、仪表等）都看作服务。Web Service 提供了一种基于服务的框架结构，但这种服务往往是永久的，而在网格上的服务往往是大量的、临时性的。所以，OGSA 在 Web Service 概念的基础上，提出了网格服务（Grid Service）的概念，用于解决服务发现、动态服务创建、服务生命周期等与临时服务有关的问题。

3．计算池模型

计算池模型把分散各地的高性能计算机用高速网络连接起来，用专门设计的中间件有机地粘合在一起，以 Web 界面接受各地用户提出的计算请求，并将之分配到合适的节点上运行。计算池模型将计算机网格限定为以下三点：计算资源共享；不把一个任务分解成 N 个子任务，而只是安排在其中一台合适的机器上运行；通过 Web 提交任务和查看结果。

20.3 网格计算的环境

本节简单介绍 Globus 的网格计算环境。

Globus 是由美国 Argonne 国家实验室与南加州大学信息科学学院合作开发的一个名为 Globus 的项目，它是一种研究网格环境中互操作的中间件技术，为科学和工程上的网格计算应用程序提供基本的支撑环境。Globus 是来自世界各地关注网格计算概念的部分研究人员和开发人员共同努力的成果，它围绕 4 种主要活动来组织，分别是研究、软件工具、试验台、应用程序。

Globus 项目发布的软件名称为 GlobusToolkit，主要面向大型科学和工程计算。GlobusToolkit 提供以下主要功能：

（1）网格计算所需的一组基本工具，包括安全性管理、资源管理、数据管理、信息服务，以及这些工具的 API。

（2）构建和编译程序所需的 C 绑定文件（头文件）。

此外，也可以使用在这些工具之上构建的其他构件。例如，Globus 提供了称为 CoG（Commodity Grid）的快速开发包，它支持诸如 Java、Python、Web Service、CORBA 等技术。

OGSA 是 GlobusToolkit 的第三版设计的原形。OGSA 的识别服务、改变和更新服务的机制有以下几个方面。

（1）接口：支持发现服务、了解服务特性和接口，从而使用服务、动态创建服务，进行生命周期管理和通知（被多个分布式服务用来相互通知相关的状态修改等操作）。

（2）宿主环境（Hosting Environment）：一个服务实际应用中的执行环境。目前，科学计算网格服务实例是以本地操作系统为宿主环境。

（3）资源管理（Resource Management）：包括资源说明语言（Resource Specification Language，RSL），描述资源管理各个成员之间交换的资源申请、资源管理成员、本地资源管理。

（4）数据管理：对数据的存储和传输进行管理，包括网格文件传输协议 Grid FTP 和数据复制管理。

20.4　普适计算概述

随着计算技术的发展，PC 和 Internet 所带来的那场信息技术革命已经变成了技术的底蕴和基础。我们手中的电子产品越来越多，在公共汽车上，可以用文曲星记单词；在商务活动中，可以使用电子笔记录会谈内容；在家里，可以使用计算机了解公司股票涨跌的情况等。真所谓是“手机、呼机、商务通，一个都不能少”，方便之余，发现要拿的东西越来越多了。但是，每天都要背着一大堆电子仪器出门，却不是一件“幸事”，有没有想过这样一幅场景：

晚上刚下飞机，饥寒交迫的希赛教育王总通过手机发送一条“我一小时后回来”的短消息给他的电子生活助理。然后，生活助理开始工作，首先得到冰箱的报告，肉和蔬菜开始解冻、但是主人喜欢的可乐没有了。于是，生活助理发 E-mail 给希赛生活中心定购了一箱可乐和一束鲜花，30 分钟后购买的物品就送到了。同时，电子厨师开始按王总所喜爱的食谱做饭菜。在王总跨进家门的前 5 分钟，他启动了热空调。现在宽大的玻璃桌上铺垫着白色的桌布，印着银色的花纹图案的餐具下垫着一块金丝边的餐巾，在满天星与蜡烛的营造映照下，晶莹剔透的水晶餐具显露着迷人的表情。王总心安理得地品尝着德国热红葡萄酒、吃着蒜香肉丝，而他不用理会发给希赛的邮件是采用移动网络还是有线网络，甚至不用知道电子生活助理在哪儿。他已经感受不到计算机的存在了。

这也是梦吗？不是，普适计算将带领人们进入这个新奇的世界。

20.4.1　普适计算的发展

计算模式在计算机的发展过程中，起着至关重要的作用。从最初的主机计算到现在的桌面计算，计算模式已经经历了两个时代。

在主机计算时代，人与计算机的关系是多对一的关系，计算机安装在为数不多的舒适的计算中心机房里，人们面对的是打孔的纸带。同时，必须用机器语言与计算机打交道。此时，信息空间与人们生活的物理空间是脱节的，计算机的应用也局限于科学计算领域。人们的许多精力花费在如何熟悉计算机上。计算机本身性能、功能的提高是人们工作的重点，而不是在解决问题上。

在桌面计算时代，人与计算机的关系是一对一的关系，计算机走出温室进入了千家万户。此时，信息空间和人们的生活有了一定的联系，计算机呈现在人们面前的是图形

界面，而人们开始忽略机器语言，开始用点击的方式和计算机交互。计算机开始应用于企业管理、金融分析等领域。解决一个问题需要人们花费许多精力在如何协调多个计算机工作，工作资源如何共享和搜集、分类资料上面。

在过去的数十年里，计算机技术有了飞速的发展，它的通信能力和计算能力更加强大、价格更加便宜、体积也变得越来越小。随着各种传感器技术和网络技术的不断进步，以及人们对网络技术的重视，将要带来计算模式的第三次革命，即普适计算（Pervasive Computing 或 Ubiquitous Computing）时代。

在普适计算时代，人与计算机是一对多的关系。同时，计算机主要不是以单独的“主机+显示器”的设备出现，而是采用将嵌入式处理器、存储器、通信模块和传感器集成在一起，以信息设备（Information Appliances）的形式出现。这些信息设备集计算、通信、传感功能于一身。不仅如此，信息设备还可以十分廉价地通过无线网络与互联网连接，并按照用户的个性需求进行定制，以嵌入式产品的方式呈现在人们的工作和生活中。或是手持的，或是可穿戴的，甚至是以与人们日常生活中所碰到的器具融合在一起，可以想象未来的微波炉可以自动下载菜谱，冰箱可以自动订购可乐等。结果是，由通信和计算机构成的信息空间将与人们生活和工作的物理空间融为一体，人们再也不需要常常坐在电脑桌前和计算机交互了。计算已经从人们的视线中消失，变得和空气、水、土壤一样重要而且无所不在，但是人们已经不大注意它的存在了。从前娇贵的计算机已经以用户为中心，默默的提供着人们所需要的服务。

普适计算的思想由 Mark Weiser 在 1991 年提出，并从 20 世纪 90 年代后期开始受到广泛关注。Mark Weiser 是这样描述普适计算的：“最具有深远意义的是那些从人们注意力中消失的技术，这些技术已经渗透到人们的日常生活中，以至于和生活难以区分”。

20.4.2 普适计算的特性

间断连接与轻量计算是普适计算最重要的两个特征。普适计算的软件技术就是要实现在这种环境下的事务和数据处理系统。

普适计算的第一个特征是间断连接，是服务器能否不时地与用户（特别是移动用户）保持联系。用户必须能够存取服务器信息，在中断联系的情况下，可以处理这些信息。所以，企业计算中心的数据和应用服务器能否与用户保持有效的联系就成为一个十分关键的因素。由于有部分数据要存储在普适计算设备上，普适计算中的数据库成为一个很关键的软件基础部件。

普适计算的第二个特征是轻量计算，即计算资源相对有限。普适计算主要用于商业用途的数据处理，通常针对移动办公的工作人员和需要经常在旅途中存取公司系统数据的职员，他们需要不受地域和时间限制地获取和处理核心系统上的数据。

20.4.3 普适计算的应用领域

普适计算按应用可以分为以下几个方面。

（1）智能环境感知技术：在人体周围环境中布置智能化的技术，而不为人所感觉到。其包含环境感知、不可见计算、智能及自然交互作用三大关键技术需要攻克。

（2）无缝的可移动性：能实现系统为其服务的主体无论是在任何位置，感受不到服务效果的变化。包括系统无缝性、业务无缝性、覆盖无缝性等多方面的领域。同时，目前流行的 IPv6、3G 技术和多业务接入技术初步开始体验无缝的可移动性带来的方便。

（3）普遍的信息访问：用户无论在何时、以任何方式访问他们认为有用的信息。现在的设备往往只能合适一种系统，例如，手机的内容就不能接入到打印机打印出来，而在文曲星中写下的日记也不能发送到邮箱里。因此，当设备进入一个新的环境时，它要能自动发现环境中可用的信息设备和服务，利用这种技术，就能较好的解决这类问题。

（4）觉察上下文计算：系统能决策在当时情景中与交互的任务有关的上下文，据此做出决策并自动提供相应的服务。在桌面系统中的觉察上下文是指文章的上下文识别，并且对上下文做出翻译或判断，这在人工智能领域已是困惑专家们很久的问题。在普适计算中，觉察上下文有了更高的要求，它指的是能识别服务主体所在的环境。例如，同样是输入“雨伞”，在三亚的海滩边，服务系统会给用户一顶太阳伞和一瓶防晒油，而不是给用户一件雨衣和一双防水靴；当用户被匪徒追赶的时候，系统会自动向警方报告用户的位置和逃生的路线，而不是给出合适用户的运动靴的品牌和价格。可以将上下文分为以下几部分。

- 计算上下文：例如，网络的连续情况、通信成本、通信带宽和附近的资源等。
- 用户的上下文：例如，用户的特性、位置、时间、附近的人员等。
- 物理的上下文：例如，温度、磁场强度、湿度、气压等。
- 上下文的历史：例如，觉察上下文计算中关键技术的获取技术。

（5）可以穿戴的计算：是一种能跟随使用者任意移动的新型计算机系统，具有可以再编程的能力。

20.5　普适计算系统的组成

一般而言，普适计算系统由如下几个方面组成。

（1）新的嵌入式系统：以嵌入式计算机为技术核心，面向用户、面向产品、面向应用，软硬件可裁减的，适用于对功能、可靠性、成本、体积、功耗等综合性严格要求的专用计算机系统。嵌入式系统主要由外围硬件设备、嵌入式处理器、嵌入式操作系统以及特定的应用程序等 4 部分组成，是集软硬件于一体的可独立工作的器件，用于实现对其他设备的控制、监视或管理等功能。新的嵌入式系统应具有的特点是：高可靠性、实时处理能力、系统识别性和交互性。

（2）系统软件：是构成普适计算基础的软件部分。它能对联网设备、物体、计算实体进行管理，为它们之间的数据交换、消息交互、服务发现、任务协调等提供系统级的

支持。它和其他分布式系统的区别是：物理集成（计算节点和物理世界的某种集成）和自发的互操作。现在的移动设备还不能做到良好的自发的互操作。

（3）普适网络：普适计算是网络计算的自然延伸，无线网络将成为普适计算的中心，而网格可能构成计算和资源的平台。现在 4G 技术相比 3G 具有更高的数据率、更强的业务质量、更高的频谱利用率、更高的安全性、更高的智能性、更高的传输质量，具有良好的覆盖性能，在多个移动网络和无线网络间无缝漫游，是一个支持 IPv6 的全 IP 网络。而网格能给人们提供快速的服务。

（4）交换模式：能实现频繁而复杂的交换方式。

20.6　普适计算的关键问题

目前，普适计算还没有太多的应用，因为它还有许多问题未能够解决，主要问题如下。

（1）发现。当新设备进入或已有设备添加新的模块时，系统如何发现它，并且和它交互。面向对象技术给予研究人员很大的启示，就是把每一个新设备或新模块看作一个对象，这样，就可初步实现动态的服务增减。但是，这需要一个全世界软硬件厂商都认可的协议的支持。

（2）感知。如何感知万事万物，即如何辨别服务主体在时间、空间、身边物资的变化。现在的摄影机、数码相机、温度计等感知仪器还不能达到“乱真”的精度。同时，还不具备分析环境的能力。

（3）分析关键因素。在判断上下文中找到关键因素。例如，判断雨伞的关键因素在于天气和温度。这需要建立庞大的资源库和情境模型。

（4）强壮性。无线通信中，模块间不可能实时地联系，这样，服务的对象很有可能有一段时间不在服务区（进入盲点），数据不可避免的丢失。在普适计算中，这应该视为正常而不是故障。

（5）微型化、可持久性。如果设备耗电量大，人们就得把精力耗费在更换动力系统上。

目前，人们还不得不坐在计算机前工作，还不得不敲击着一个个的字符来表达思想。但是如果有一天，人们感受不到计算机的存在，而每项工作却都有它们默默的支持，同时，人们从中得到了很大的便利。这样，网格计算和普适计算的时代就到来了。

本章参考文献

[1] 孙培德. 网格计算的研究新进展.计算机工程与应用，2003

[2] 邓静，王帮海，徐建哲. 网格实例——Globus 的研究与探讨. 计算机应用，2003

[3] 陈萍，余华山，王彬. 网格计算环境 Globus 介绍. 计算机应用研究，2003

[4] 胡敏，顾君忠. Globus 网格体系结构及其服务的实现. 计算机工程，2003

[5] 徐志伟，李伟. 织女星网格的体系结构研究. 计算机研究与发展，2002

[6] 徐志伟，李晓林，游赣梅. 织女星信息网格的体系结构研究. 计算机研究与发展，2002

[7] 杨新刚，李晓林，唐宁九. 织女星企业信息网格资源接口模块设计与实现的研究. 计算机应用研究，2003

[8] 徐光祐，史元春，谢伟凯. 普适计算. 计算机学报，2003

[9] 王玉玺，沈为民. 可穿戴计算机的研究发展现状. 哈尔滨师范大学自然科学学报，2002

[10] 隆升. 可穿戴计算机. 现代通信，2002

[11] 刘鹏. 网格应用现状与分析——从应用看网格. http://www.chinagrid.net

第 21 章　云计算与 SaaS

网络大大扩展了计算机的计算能力和应用范围，尤其是随着互联网的出现。互联网使得基于计算机的服务提供方与使用方之间能够进行友好度和扩展度都更优的充分交流。人们很早就提出和实现了基于网络的多台计算机的协同技术，如分布式技术、服务器集群技术、负载均衡技术、Web Service 等，在互联网的基础上对这些技术进行扩展，再加入一些创新，基本就构成了现在的云计算。而软件即服务（Software as a Service，SaaS）则是基于互联网的服务提供、软硬件资源租赁、数据存储、安全保障等服务的商业应用。云计算是一种体系结构和设计方式，而 SaaS 则是云计算的一种商业应用。

21.1　云计算概述

2006 年，以网上书店闻名的 Amazon 公司推出了 S3（Simple Storage Service），从它最简单的功能描述上可以粗略得知，S3“类似”于已存在多年的网络存储服务，例如，虚拟主机、网站空间等。亚马逊公司还推出了 EC2（Elastic Compute Cloud）服务，EC2 服务实现了计算资源的租赁。借助这两大服务，可以在不需要任何前期投入的情况下，设计一个迎合各类需求的计算平台解决方案。而提供这些服务的理论支撑，就是所谓的云计算。

21.1.1　云计算的概念

除 Amazon 以外，Google、Microsoft、IBM、EMC、SUN 也都提供一些基于云计算的服务。例如，Google 所有的服务都是基于其云计算服务器群的；Microsoft 也启动了 Office Online 服务，这一服务的想法之一就是将中小型企业的办公系统迁移到 Microsoft 的云计算服务器中，例如，Exchange 迁移到 Exchange Online；IBM 在 2007 年底启动了 Blue Cloud 的云计算计划。

虽然从表面看来，云计算被炒得如火如荼，但实际上到目前为止，连云计算的概念都没有达成实质性的共识。在 Microsoft 看来，是用户将自己 Exchange 和 Office 都过渡到其 Online 平台；在 IBM 看来，是用户将自己的数据与信息中心置入“蓝云”的安全保障之下；在 Amazon 看来，是以类似虚拟机的方式向用户提供计算资源、数据存储，甚至开发环境迁移到 S3 和 EC2；在 SaaS 软件厂商看来，云计算则是用户将自己的业务逻辑和运营数据存放在其 B/S 架构的远程系统中。

英文维基百科对云计算机做出了如下的定义：云计算是一种基于 Internet 的计算机

开发和使用技术，它是一种动态可扩展的并通过 Internet 以服务的方式提供虚拟化资源的计算方式。对于提供支持的技术基础设施“云”，用户无需对其了解，也不需要相关使用经验，更不用对其进行控制。

中文维基百科对云计算给出了如下的说法：云计算是分布式计算技术的一种，其最基本的概念，是透过网络将庞大的计算处理程序自动分拆成无数个较小的子程序，再交由多部服务器所组成的庞大系统经搜寻、计算分析之后将处理结果回传给用户。

总结一些对云计算的概括，并对云计算的一些具体应用进行剖析，可以对云计算的定义进行如下的归纳：云计算是一种基于并高度依赖于 Internet，用户与实际服务提供的计算资源相分离，集合了大量计算设备和资源，并向用户屏蔽底层差异的分布式处理架构。

云计算之所以能够得以应用，一个重要的原因之一就是 Internet 的高度发展，只有全球范围内的大规模网络的出现，才使得面向全球的计算资源服务能够被随时随地的方便调用。在云计算中，“云”以及其真正的使用者，很多情况下是分布在不同地区的，用户通过 Internet 来实现对云计算资源的使用和利用。“云”也有可能因为各种因素而分布在不同的地理位置上。通过对这些云端的集成，屏蔽其底层差异，使得用户获得一致的服务体验。

希赛教育专家提示：为了满足应用，云计算通常需要大量的计算机作为支撑，从目前的云计算应用上讲，其组成的计算机的数目都是非常大的。例如，Google 为了支撑其大量的用户搜索请求以及自身的 Reader、Calendar、Picasa、YouTube 等应用，使用了大约超过 25 万台服务器。

21.1.2　云计算的应用

总的来讲，云计算是一种大量服务器的组成架构，其提供的计算资源并不能直接的给用户使用，而是通过其他的方式，例如，向用户提供搜索、存储、相册、Blog、科学计算等应用服务的方式来展现其魅力。在目前，云计算已经被应用到以下几方面。

（1）存储服务：例如 Amazon 所提供的 S3，就是一种向用户提供存储服务的云计算应用。

（2）搜索：各大搜索引擎公司（如 Google 等）为了满足用户的需求，并提供良好的用户体验，都使用了大量的服务器，组成服务器群，把用户的请求进行拆分、执行、返回。

（3）科学计算：小型团队在实验或项目必须的情况下，必定会有大量的计算需求，但无论是购买设备，还是租用大型计算机，都将有不菲的费用，而通过购买云计算的资源（如 Amazon 的 EC2 服务），搭建需要的平台，基本可以在零前期投入的情况下来满足相应的计算需求。

（4）SaaS：通过利用 B/S 技术，将企业的业务逻辑和数据都置于云计算的服务器群

中，以适应中小企业的低成本满足应用需求的要求，如 Salesforce.com 等。

云计算的应用场合较广，一般地，当有以下需求的时候，就可以考虑使用云计算服务。

（1）短时间内陡增的计算需求。

（2）零成本的前期投入，并且总体拥有成本（Total Cost of Ownership，TCO）较优。

（3）在充分相信云计算服务提供商的情况下的数据安全性需求。

（4）没有足够的服务器管理和运维人员。

（5）在终端设备配置较差的情况下完成较复杂的应用。

当使用云计算服务时，一般都可以达到前期成本的零投入，短时间内在云计算环境中组建一个满足大规模计算需求的虚拟服务器或虚拟服务器集群。而且，用户不需要专门配置维护人员，云计算服务的提供商也会为数据和服务器的安全做出相对较高水平的防护。由于云计算将数据存储在云计算的云端中，业务逻辑以及相关计算都在云计算的云端完成，因此，终端只需要一个能够满足基础应用的普通设备即可。

21.1.3 云计算机的特点

云计算作为一个新兴事物，虽然在理论和应用上都没有得到一致性的共识，但也可以归纳出以下几个方面的特点。

（1）集合了大量的计算机，规模达到成千上万。一方面，大量的计算机可以提供强大的整体计算能力；另一方面，整体管理还可以降低管理和维护成本，通过对计算机运行环境的优化，提高单台计算机的服务寿命。

（2）是多种软硬件技术结合的产物。云计算是一些已有技术的结合体。在云计算的组织结构中，使用到了诸如分布式技术、负载均衡技术、服务器集群技术等；在基于云计算的应用设计中，还会用到 B/S、Web Service、SOA 等；在硬件组织和机房建设中，又会使用到一些现已成熟的冷却、通风、布线等技术。

（3）对于客户端设备的低要求。通常，云计算的客户端系统只需要满足能够运行一个浏览器的要求即可。而且，云计算的客户端是多样的，可能是一台 PC，也可以是一部手机。客户端只需要将相应的数据展现给客户，并对用户的输入进行收集和提交即可，业务逻辑中的大部分都将转换到云计算服务器上，数据也将存储在“云”中，在商业的 SaaS 应用中，大部分的客户端都是浏览器。当然，有些情况下需要安装一些插件。

（4）规模化效应。云计算的服务器是大规模的，用户也是大规模的，这使得管理与维护都得以集中，不仅降低了服务器的维护成本，还使得软硬件资源得以最充分的利用。当然，这在很大程度上也加深了灾难的蝴蝶效应，一旦云计算的关键设施（如服务器“云”）出现问题，甚至数据丢失，或网络发生异常，对于客户的影响将是致命的。

21.1.4　云计算与网格计算

与云计算一样，网格同样能够提供巨大的计算能力，但二者有着以下几方面的不同。

（1）网格是利用分布在网络中不同机器的闲置资源来进行计算，而云计算一般是将一些计算机进行集中，再进行相关架构；网格计算在一般情况下，并不专门的添加各种硬件，而是对现有资源进行充分利用，而云计算通常都是重新组织计算机。

（2）网格计算是分布式计算的一种，主要是在现有的异构平台下，开发一些相应的网格计算软件来间歇性的运行，以实现计算目的；而云计算是一套整架构，是将计算资源进行的软硬件集中。

（3）网格计算通常是为了完成某一特定任务而以项目为单位进行的计算，而这个任务通常都是计算量非常大的，类似于使用“蚂蚁搬家”战术；而云计算的应用则广泛得多，通常云计算都是为了提供某种服务，如搜索、存储等。

（4）网格计算的各种硬件计算资源通常分属不同的所有者，这些所有者处于某种目的，或公益性的贡献自己的计算时间，而云计算的运营者通常都是某个组织或机构；网格计算大多是公益性的项目，而云计算从一出现，就开始积极走商业化道路。

（5）网格只在数据传输时需要使用网络，而且不一定是互联网；而云计算在进行服务应用时，通常需要进行实时的数据传输。

（6）网格计算通常只是为了单纯地进行计算，而云计算则更丰富，目前，云计算除了计算资源，还涉及到存储资源的利用。虽然商业化的网格计算会涉及到存储，但这时候，可能称其为分布式存储更为适当。

（7）网格是将计算任务分解成小部分，再分别执行；而云计算是将需求转化为服务，接着将服务转变为应用计算或存储需求，再将其分给云中的各台计算机。

另外，网络计算并不仅限于 Internet。例如，企业为了更好地利用内部的闲置计算资源，也会开发一些网格应用，来补充集中性的服务器计算资源的不足。

21.2　云计算的架构

云计算的架构从总功能上，可以分为 6 层，从上到下分别是客户层、服务层、应用层、平台层、存储层、基础设施层。

1．客户层

客户层是云计算的最终用户所接触的一层，而且，在用户看来，另外 5 层都是不透明的，用户不需要知道自己的请求最终由哪些计算机怎样去完成，只需要将自己的任务提交给“云”即可。客户层的要求是很低的，大多数情况下，只需要满足运行浏览器的要求即可。云计算中的数据存储和业务逻辑在客户层中都被弱化。甚至有人设想，某一天，计算机不需要多强大的功能，只需要能够打开浏览器即可，其他的一切都在“云”

中进行。

客户层的设计一般都是跨平台的，可能会运行在移动设备、瘦客户端，以及普通的胖客户端。而无论用户使用何种设备，都可以完成一些主要服务和功能的应用。而到服务成熟以后，用户无论是使用手机还是普通的 PC，都不会感觉到太大的差异（除部分硬件条件的限制外）。

2．服务层

服务层主要是将应用以服务的形式提供给用户，也是云计算中与用户进行直接性交互的一层，该层将云计算的其他层进行屏蔽。一方面，客户层通过服务层利用云计算的各种资源；另一方面，云计算将所有资源以服务的形式进行封装。

当然，服务层并不仅仅针对直接用户，也可以为其他云计算提供服务，服务层是云计算的输入输出接口。各种应用，都可以以服务的形式进行封装并运行在云计算上。

3．应用层

应用层主要运行直接性提供服务的应用程序，它将云计算的计算资源转化成实际的服务，以实现对计算资源的封装。例如，SaaS 的服务端就是以应用程序的形式运行在云计算服务器上。

4．平台层

平台层主要是提供应用层程序运行的环境，并对相关的计算资源进行调配。例如，在 Amazon 提供的 EC2 服务中，将计算资源以类似虚拟机的方式提供，在使用 EC2 资源之前，需要对其进行系统的安装和配置，Amazon 也提供一些较常用的镜像，供用户进行使用。

5．存储层

存储层主要实现存储资源的整合以及分配，云计算除了产生巨大的计算压力外，还需要对大量的数据进行临时或永久性的存储。而且，为了安全考虑，这些数据通常都按照一定的策略进行安全保障。云计算的服务对象众多，还必须对这些数据进行隔离或有条件的共享，这使得存储资源的分配和管理变得更加复杂。

例如，在 Amazon 提供的服务中，将云计算的永久和临时数据分开管理和存储，永久数据存储由弹性块存储（Elastic Block Storage，EBS）和 S3 来完成，而 EC2 提供的存储空间，仅在实例运行期间有效，当停止后，所有数据将丢失。

6．基础设施层

云计算的基础设施相对概念较广，一般可概括为云计算的计算资源来源，可能是一个或多个服务器群，甚至也可以是一些由网格计算技术组织的计算机资源，还有可能是其他的云计算提供方。

希赛教育专家提示：云计算作为一种计算资源集中和使用的方式，符合未来日益丰富的互联网应用的计算需求，也为组织计算资源提供了一种新的解决思路和方案。

21.3　SaaS 概述

SaaS 是目前较为热门的一个应用，尤其是在中小企业的信息化中，在信息化项目的预算有限的情况下，可以有效地保证投资的安全性，而且后期不需要专人维护，对于用户来讲，只需要接受相应的服务。当然，目前也存在一些问题，诸如数据的安全问题、服务的客户定制问题等。

21.3.1　SaaS 的定义

企业信息化的最开始，是企业提出需求，由软件开发公司或企业的 IT 部门进行定制开发，从业务导出需求，再进行功能划分，最后进行实现。到了后来，随着需要信息化的企业越来越多，相关的开发经验也越来越成熟，便出现了信息化产品，诸如以功能进行集中的 ERP、CRM、SCM 系统等。而随着软件架构从 C/S 到 B/S 的过渡，使得企业的信息化业务在浏览器中即可完成。B/S 的广泛应用，从某个方面来讲，催生了 SaaS 的产生和推广。

不同的企业有不同的业务和流程，这也使得具体的需求会有所不同。但是，在 CRM、HRMS、OA、SCM 等方面的需求，对于多数中小企业来说，都有着很大程度上的相似性，需求相同意味着软件可以进行较深的复用，一些信息化服务提供商很早就注意到这个问题，并已经开始了复用。

需求的复用，B/S 三层架构的表现、逻辑、数据相分离，Internet 的大规模应用以及性能的提升，使得 SaaS 在技术上具备了产生的条件。

SaaS 是一种以互联网为基础，将应用和软件以服务的方式提供的软件运营模式。对于用户来讲，服务和数据就是其信息系统的全部。系统的管理和维护将被集中，由 SaaS 运营商来承担相关工作，SaaS 的运营商通常还会是软件的开发商。

SaaS 的软件运行在由 SaaS 运营商统一管理的服务器上，也有可能是运营商搭建或租用的云计算资源，软件的升级、维护、管理都将由运营商负责。用户只需要使用浏览器打开网页，输入账号，即可开始使用。而且，服务的使用没有具体的地点和时间限制，只要具备一个连接到运营商服务器的网络即可，通常，这个网络就是 Internet。在成本和预算方面，除了有可能存在的相关咨询费用外，用户前期基本可以做到零投入。一般的 SaaS 运营商都提供一段时间的试用，再确定系统是否符合要求。这也大大保护了用户的投资。SaaS 运营商还提供产品的定制，即在 SaaS 中为企业做一些业务需求的定制，以更适合企业本身。投入使用后，费用也是按使用的规模、功能以及时间来确定的，这也降低了企业的 TCO。

21.3.2　SaaS 的特点

SaaS 是最近几年才出现的一种 Internet 软件运营和销售模式，与传统的软件运营模

式相比，它有以下几个特点。

（1）高度依赖 Internet。虽然在理论上，只要用户与 SaaS 运营商的服务器有网络连接就可以完成，但实际上，这个网络连接通常由 Internet 来完成。SaaS 的产生原因之一就是 Internet 的发展，Internet 的普及率，以及速率、稳定性都得到了一定程度的提升，这对于不间断要求较高的商业应用尤其重要。

（2）软件架构几乎都基于 B/S。B/S 的一个重要特点就是客户端的标准化，使得其部署非常简单、方便，甚至基本不需要部署，通常的计算机都能完成这个任务。B/S 还带来了表现、逻辑和数据的分离，这使得服务的提供能够更简便。

（3）TCO 最优。几乎为零的前期投入，按功能、规模和时间取费的收费策略，无论是对于投资保护，还是降低成本，都具有决定性的作用。尤其是 SaaS 运营商提供的免费试用和功能定制，更为降低 TCO、避免浪费提供了更多的保证。另外，SaaS 免于系统管理和维护，也节省了企业的人力运营成本。

（4）多用户并行于一套系统。SaaS 之所以能够使得 TCO 更低，原因之一就是多个用户的资源共用。这包括服务器计算资源、网络带宽，甚至是程序和数据级的共享，例如，多个用户使用同一个系统，将数据存放于同一个数据库中等。

（5）集中的系统管理与维护。B/S 中的业务逻辑层和数据层被转移到 SaaS 运营商的服务器上，由其进行集中系统管理与维护，以及软件产品的修改、升级等。不仅提升了系统管理和维护的水平，便于软件系统的更新与升级，也为企业降低了相应的 IT 运维部门的人力需求。当然，集中的管理模式也会造成灾难，这加大了系统管理与维护的安全压力。

（6）安全隐患。安全隐患可能来自于 SaaS 运营商内部和外部，甚至 SaaS 软件的其他用户。有意或无意的破坏都会有非常大的影响，尤其是在多个企业数据被集中的情况下，更容易产生灾难的规模效应。虽然数据存储在企业内部也会有安全问题，但大多数中小企业都不习惯“将自己的鸡蛋放在别人的篮子里”。安全隐患的顾虑，有时也来自信任问题，企业的数据都是其重要财产之一，尤其是客户资料和财务数据，存放在企业外部的服务器上，难免会让人对 SaaS 运营商产生信任问题。

希赛教育专家提示：规模化经营是一个行业发展的趋势，软件行业也不例外。SaaS 使得软件由一种产品转变成了服务，将需求的满足从系统的管理和维护中剥离，这都使得软件应用和信息化能够真正地走进信息化时代。当然，其中的问题也必须正视，而且安全和信任问题，是 SaaS 发生的巨大障碍。

21.3.3 SaaS 与 ASP

应用服务提供商（Application Service Provider，ASP）是 20 世纪末期的一个信息化术语，其主要观点是，企业处于降低成本的考虑，将其自身的服务器或信息系统交给 ASP 运营商托管，由运营商来负责系统管理和维护。在当时，ASP 应用并没有流行起来。不

少对 SaaS 质疑的人称“SaaS 就是 ASP 新瓶装旧酒”。

ASP 出现的时代，是在 20 世纪末期，当时存在着技术和环境上的一些不利因素。20 世纪末信息化的主要架构是基于 C/S 的，C/S 系统中的客户和服务器两端之间往往有着很多的交叉，部分计算需要在客户端上进行。客户端还需要专门部署，这都使得客户和服务器分布在互联网的两端会产生很大的不便，其他相关的支撑技术的不成熟也对此产生了影响。在 20 世纪，互联网无论是速度、普及率还是稳定性，都不能很好满足实时商业应用的需求，VPN 也无法解决 C/S 对速度和质量的要求。而现在，这些问题都基本已经解决。

ASP 服务通常是一个客户一套系统，甚至独享服务器资源。而 SaaS 在大多数情况下，都是多个用户，甚至成百上千的用户共享包括软硬件甚至数据在内的各种资源，这也使得 SaaS 的成本远远低于 ASP 的成本。

SaaS 的软件以服务的方式提供，而 ASP 则仍然是以软件的方式来提供，SaaS 的软件通常已经被开发完成，或大部分开发工作已经完成，只需要进行一个配置或相对较小工作量的定制即可。而 ASP 则不一定，虽然其仍然可能是以产品为对象，但是其软件服务化倾向要小得多。

SaaS 与 ASP 有着一定的区别，相对来讲，二者都尝试让客户更容易实现信息化，但 ASP 则更像是帮助客户解决因聘请专门的 IT 运维人员而带来的人力成本上升问题。SaaS 是 ASP 在新技术以及新环境下的发展，但把二者混为一物，也是一种久妥的说法。

21.4　SaaS 应用的问题

SaaS 在近几年得到了较多的应用，也产生了很多 SaaS 运营商。例如，CSAI 顾问团就在 CSAI.cn 中提供了一个 SaaS 模式的项目管理平台。但有很多问题仍然困扰着 SaaS 的发展。其中，尤以信任以及用户的观念转变和安全最为关键。

21.4.1　SaaS 的信任危机

在 SaaS 应用中，企业的数据和业务逻辑都被转移到 SaaS 运营商的服务器中，而企业的财务和客户数据，不仅是其重要的资源和商业秘密，后者还是一种其相对于竞争对手的信息和资讯优势的体现。

信任危机主要体现在以下几个方面：

（1）运营商能够确保自己的数据不被竞争对手盗取吗？

（2）如果运营商将这些商业秘密泄露或出卖，该怎么办？

（3）当有一天不再使用该运营商的服务时，这些数据如何平滑过渡？如何确保运营商不对即将转移的用户施压？

另外，还有因安全问题而造成的信任危机，在 21.4.2 节将对此进行讨论。除了安全

问题，还有一些阻力，例如，很多企业根本不想让除了自己以外的任何人知道自己的财务信息，就算是企业内部，也只是有限流通。

以上各种问题都是异常关键的，也都是非常棘手的。这不仅涉及一些行业规范，还有技术上的一些问题。笔者认为，综合考虑信任危机，可以从以下几个方面着手健全安全机制。

（1）完善和规范相应的法律法规，以期对可能造成的后果的行为进行威慑。

（2）制定相关的行业准则，做好行业准入及监督工作，并建立一个中立的企业信用档案。

（3）将 IT 监理引入到 SaaS 运维行业，做好事中控制，并进行事后审计。

（4）对于企业数据的安全保障，引入保险机制，以期对已经造成的损失进行补偿。

信任危机是不可能彻底消除的，只要有分工与合作，总会产生信任问题。目前所做的一切努力，只能缓解信任问题，不可能完全消除。而且，信任是一种“瓷器”，创造和积累过程极其困难，但只要有一点外力，即可前功尽弃。

21.4.2 SaaS 的安全问题

安全问题是爆发信任危机的根本原因。在 SaaS 应用中，安全问题主要有以下几方面：

（1）客户端的安全。当客户端出现问题，例如，被人恶意安装键盘记录程序或木马程序，通过记录其操作，来盗取企业数据。对于客户端的安全问题，可以通过杀毒软件和防火墙，并配以良好的计算机和网络使用习惯，制定较为严格的办公用计算机使用规则，即可一定程度上保证客户端的安全问题。

（2）网络传输的安全。如何保证数据在网络传输的过程中不被窃听、篡改。要防止网络传输的安全问题，通信时使用传输层的 SSL 协议来进行保证，并在通信中使用一些加密和数字签名手段来防止数据被窃听、冒充和篡改。

（3）服务器的安全。这是整个安全工作中较为复杂的一环。由于服务器上存储着大量的用户数据，也是安全工作中最为重要的一环。服务器的安全比较复杂，可以从以下几个方面来进行保障：

- 采用适当的备份策略，防备应用可能存在的数据丢失隐患。数据丢失的危险可能来自数据误删除或恶意的故意删除、服务器存储设备遭遇毁灭性打击等。这可以通过相应的备份策略来进行预防。备份会导致管理和维护成本的上升，但 SaaS 一般都具有较大的用户基数，因此，对于某一用户来说，可以在较少的投入上得到较大的安全保障。
- 对于服务器的控制，尤其是关键数据的读写，可以采用非对称加密技术，并让用户掌握其私有数据的相应密钥，使得对于这些数据，在没有客户允许的情况下，运营商也不能够访问。但这又会增加对用户进行相应安全培训的工作量，并增加

TCO，而且，势必会提高系统的复杂度，降低系统的性能。

- 规范服务器内数据的共享和隔离机制。由于 SaaS 系统中多个用户共用服务器，甚至多个用户运行在一个服务实例之上，这就使得数据的共享和隔离机制变得非常重要，除了有意的安全攻击，不合理的数据共享也会造成一些不必要的麻烦和损失。

安全问题很多时候都是由于人的因素而造成的。对于 SaaS 来说，用户自己的员工、运营商的相关人员，都是进行安全教育的重点对象。除了进行技术上的培训，还需要从思想角度上防止无意识的信息泄露，甚至是被其他因素引诱的泄密。

希赛教育专家提示：安全保障是一个系统工程，其保障级别与投入有很大关系。从成本角度考虑，选择合适的安全级别是很重要的。为了无畏的安全进行投资是浪费，而将安全寄托在运气上却又是不重视的表现。

21.4.3　SaaS 带来的观念转变

SaaS 让软件以一种无形的服务的方式提供给用户，与过去的模式不同，用户信息系统没有有形的软硬件，甚至会带来花钱只买来一个网站的印象，并让很多企业陷入了无法预期的投资。而 SaaS 的软件实施，不仅速度快，而且对于企业来说，是轻量级的。这也会给用户带来一些错觉，甚至会直接导致不重视，让建成的信息系统处于成而不立的尴尬状态。

在 SaaS 实施的过程中，需要让用户和运营商进行以下几个方面的观念转变。

（1）软件不是企业的最终需求，企业信息化的最终目的是使其需求得到满足，而不管是一种什么方式，SaaS 就是以服务的方式来满足企业需求的软件经营和管理模式。

（2）鸡蛋并不总要放在自己的篮子里。只要有足够的保障使鸡蛋安全，并不一定要把它拿在自己的手中，而且专门保管鸡蛋的篮子，总是要比放在杂货箱里更安全。SaaS 其实就是将安全保障集中化，让多个用户分摊高成本和技术要求的费用。

（3）每个用户都是一份责任，用户越多，运营商更要注意自己信用的积累，也只有如此，客户才能够更多，成本才能够更低，自身才能使其更长远的发展。

观念转变是信息化中最艰难也是最重要的过程。企业信息化的成败，很多时候都与使用者对新系统的看法有很大关系。SaaS 尤其如此，SaaS 革命性变化使得这种转变更加困难，这也是当前 SaaS 推广中较为关键的因素。

21.5　SaaS 系统设计

SaaS 从整体上讲，是以 B/S 为架构，因此，其架构也可划分为数据访问层、业务逻辑层和表现层。只是与传统的搭建于企业内部局域网或外延网的基础上不同，连接 SaaS 系统的表现层和业务逻辑层之间的是 Internet。也正是如此，在进行 SaaS 系统设计时，

必须考虑到安全性问题，以及数据传输的速率和稳定性有可能会带来的各种影响。另外，SaaS 系统的用户数目，往往是较大的，不同的用户又要针对需求进行相应的定制，这都是在进行 SaaS 系统设计时必须面对的问题。

21.5.1 多租户系统设计

SaaS 系统中的多用户，术语上称之为多租户（Multi-Tenancy 或 Multi-Tenant）。当多个用户使用同一系统时，对于多租户的数据进行共享和隔离，就成了一个非常关键的问题。目前，SaaS 系统设计中用到的数据隔离共享策略如表 21-1 所示。

表 21-1 多租户的数据隔离共享策略

类 别	优 点	缺 点
独立数据库	多租户对系统设计的影响最小；可以较简单地实现对于用户的系统定制；恢复性维护更方便，隔离级别最高，安全性高	资源消耗更多，成本最高
独立数据架构	能在共享数据库的情况下，一定程序上满足用户的数据逻辑隔离要求	支持用户数量有限
共享数据架构	支持用户数多，成本最低	安全性低，针对某用户的恢复备份较为复杂；隔离级别最低

对于共享数据架构的模式，需要特别注意安全问题，这时的安全隐患还来自同一数据库内其他用户有意或无意的信息泄露。选择的数据隔离方式，主要取决于用户的需求，这也直接影响到用户的取费。

对于多租户，还需要考虑用户的管理问题，不同的员工有不同的账号，而对于不同企业的员工，还需要进行分类管理。对于企业的数据，也需要设置相应的访问权限。

21.5.2 可配置性

通常情况下，不同的企业都有不同的业务流程，而对于不同的流程，又会有不同的需求，SaaS 系统虽然是以服务为基础来满足企业的功能需求，但仍然需要一定程度的客户定制工作。SaaS 系统的定制工作，通常就是其所谓的可配置性。对于不同的用户需求，仅通过按需而行的配置来完成。

对于系统的可配置性，一般包括以下几个方面。

（1）数据可配置：需求的不同，最终是数据的不同。数据可配置有不同的策略。目前，有如表 21-2 所示的三种策略。

表 21-2 数据可配置的实现策略

类 型	描 述	优 点	缺 点
定制字段	通过新增数据库字段满足用户的不同需求	简单，易行	影响数据库结构，性能低下。基本不适合多用户的情况

续表

类　型	描　述	优　点	缺　点
预分配字段	数据库设计时，设立无含义的预分配字段	相对易行	可设立字段有限，且对于某一用户来说，还是会造成一种程度上的浪费。同一字段，对不同用户有不同意义
“名字一值”对	定制字段，使用配置元数据表和扩展数据表来将增加字段的横向扩展转化为增加记录的纵向扩展	可扩展性好、灵活、效率高，理论上定制字段可无限增加	设计复杂

（2）处理流程可配置：SaaS 中多个用户之间除了要进行数据隔离，还需要注意的就是处理流程（工作流程）的隔离，这主要是通过工作流系统的设计来完成。

（3）功能可配置：SaaS 推广着重强调的一点就是“按需使用，按需付费”，功能可配置是 SaaS 能够满足多个用户需求的重要原因。在进行 SaaS 系统设计时，一方面要注意进行功能的分解，之后，还需要注意进行功能的再打包。

（4）界面可配置：界面可配置相对简单，主要涉及页面元素布置、显示内容的控制、各种功能菜单及管理菜单的配置等。

以上各种可配置中，界面可配置给用户的体验影响最大，但作为基础的数据可配置最为重要。具体操作，需要根据 SaaS 系统设计的目标进行合理的规划和选择。

21.5.3　离线应用

SaaS 高度依赖于 Internet，而 Internet 连接有时候却会成为一个不可控的因素，例如，网络异常甚至断开。或有时候必须切断网络，如当局域网内大规模爆发病毒时，会实行断网隔离杀毒，或是在飞机上。而且，SaaS 运营商的服务器仍然有宕机的危机。这都会很大程度上影响 SaaS 系统的使用。SaaS 离线应用的设计在这时就显得非常重要了。

离线应用需要面对的问题主要有离线情况下的 SaaS 系统的本地使用，离线应用中产生数据的存储、网络恢复后的数据同步及冲突解决等问题。

如果使用传统的思路去解决离线应用问题，可以在本地架设一个备份服务器，通常情况本地服务器只是与 SaaS 运营商的服务器中的本企业数据进行备份性质的同步，当离线后，客户端的连接转向本地服务器，网络恢复后，再将本地服务器与远程服务器同步。这种方法较为全面，基本不会产生因离线而带来的过大不便。但是，该方法需要专门在企业内部进行服务器部署，大大增加了维护和管理的成本。而且，SaaS 运营商在对软件进行升级时，也会增加一些升级成本。

另一种方法是使用本地的小型文本数据库进行数据存储，如 txtDB 和 SQLine 等。当网络恢复后，再将数据进行同步。这是一种轻量级的解决思路，易于实现，成本较低，而且无需专门搭建服务器，只需要在本地运行即可。

数据本地存储会直接影响到网络恢复后的数据同步，同步过程中最常见的问题之一就是冲突。离线后，很可能会有多个用户对同一数据进行操作的情况。例如，对于一个存量为 500 的货物，离线期间，有两个用户都进行了出货 300 的请求，这其中肯定至少会有一个用户的需求不能满足，具体是哪个用户，就必须对这两个请求进行标识，一般使用时间戳加版本号的方式来完成标识，或对操作进行优先级的设置，并在网络恢复后进行相关处理。

当网络恢复后，需要对本地数据和 SaaS 运营商服务器的数据进行同步。除了利用一些具体的标识和优先级进行有判断的更新服务器数据外，还要对离线应用中的一些冲突性操作进行鉴别，并通知相应的用户具体的执行情况。

21.5.4 成熟度模型

根据 SaaS 的实现和服务方式，一些业内人士提出了 SaaS 架构的成熟度模型。Gianpaolo 和 Fred Chong 根据 SaaS 的可扩展性、多租户模式以及可配置性，将 SaaS 架构的成熟度模型分为以下 4 个阶段，如图 21-1 所示。

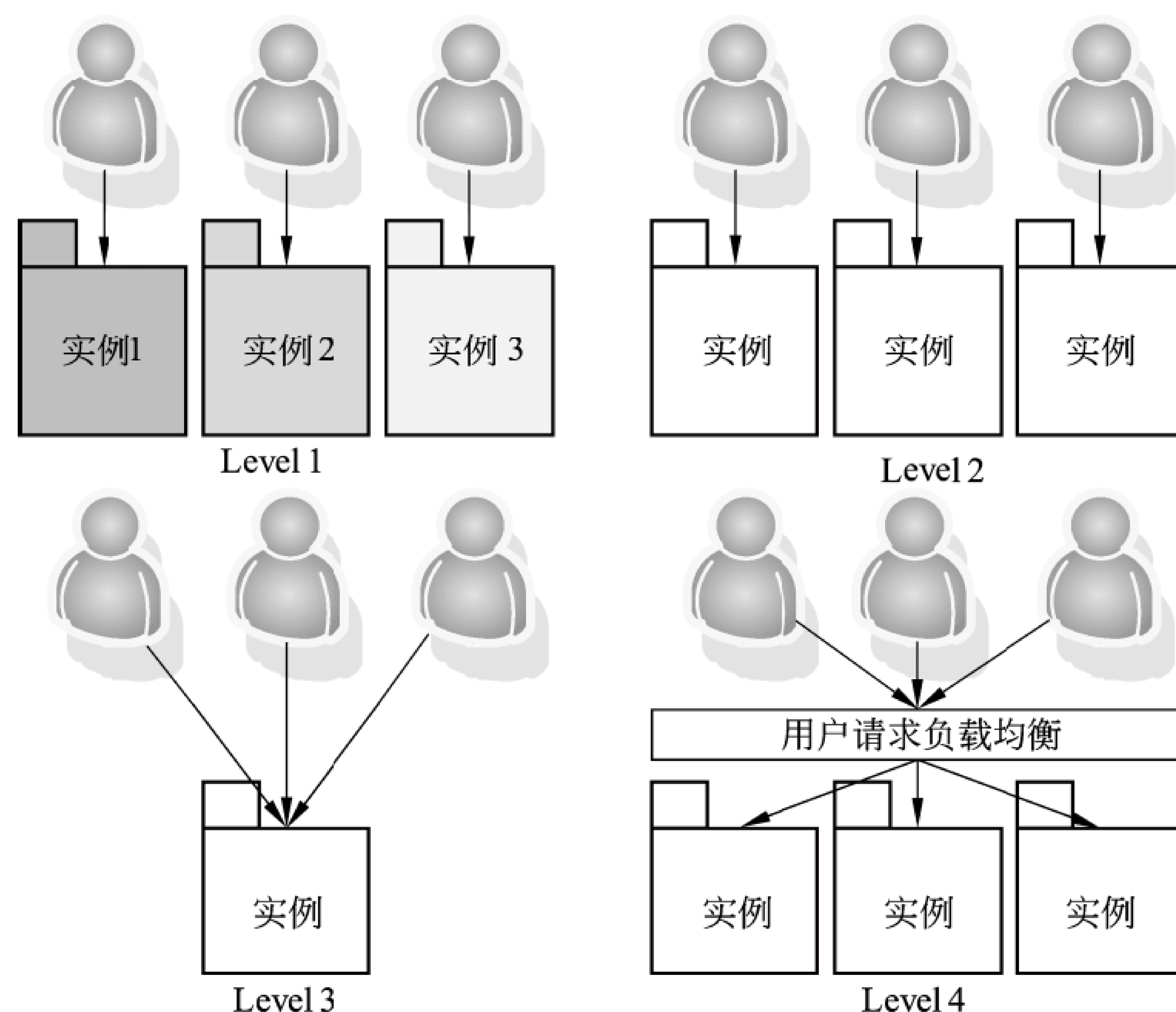


图 21-1 SaaS 架构成熟度模型

（1）Level1：为用户定制，每个用户拥有不同版本的实例。

（2）Level2：系统可配置，各个用户运行同一版本的不同实例。

（3）Level3：多租户下的可配置，多个用户运行同一个实例。

（4）Level4：具有伸缩性的多租户可配置，多个实例运行，但用户的请求，通过一个负载均衡机制来分配到各个实例。

Dharmesh Shah 提供了另外一种成熟度划分方法，他将成熟度划分为 5 个层次，分别如下。

（1）Level0（混乱级）：是一个正准备提供 SaaS 服务的企业所走的第一步，一切都只是开始和尝试，有着很多的不规范。

（2）Level1（有控制的混乱级）：相当于 Gianpaolo 模型的 Level1，已经开始尝试规范。但对于运营商而言，新用户的增加就意味着需要重新对 SaaS 系统做出一个修改。

（3）Level2（纵向增长的多租户级）：相当于 Gianpaolo 模型的 Level2，SaaS 系统已经开始趋向于成熟。对于新增加用户，不需要修改和增加代码，只需要对系统进行一些配置即可满足需求。

（4）Level3（横向增置的多租户级）：相当于 Gianpaolo 模型的 Level3，无论是新增用户还是新增功能，都可以达到可配置和可伸缩的功能和性能需求。

（5）Level4（乌托邦）：相当于 Gianpaolo 模型的 Level4，在这种情况下，新增用户对于 SaaS 运营商来讲，已经不存在技术上的障碍，运营商只需要适时地增加服务器数据即可。

本章参考文献

[1] 维基百科. Software as a service. http://en.wikipedia.org/wiki/Software_as_a_Service

[2] 月光博客. 云计算给 SaaS 的机遇. http://www.williamlong.info/archives/1371.html

[3] Rob Chappelle. Build a marketplace with the eBay SDK and Web services, Part 1. http://www.ibm.com/developerworks/webservices/library/ws-buildebay

[4] CSAI.cn. SaaS 之最佳客户体验. http://cio.csai.cn/zt/saas/

[5] 德赛网. SaaS 行业数据. http://www.dosaas.com

[6] Dharmesh Shah. SaaS Architecture and The SaaS Maturity Model.
http://onstartups.com/home/tabid/3339/bid/3629/SaaS-Architecture-and-The-SaaS-Maturity-Model.aspx

[7] Gianpaolo. SaaS Simple Maturity Model. http://blogs.msdn.com/gianpaolo/archive/2006/03/06/544354.aspx

[8] Udi Danhan. Microsoft Updates Software-as-a-Service Reference App LitwareHR with S+S. http://www.infoq.com/news/2007/11/litware-v2

[9] 叶伟. 互联网时代的软件革命——SaaS 架构设计. 北京：电子工业出版社，2009

[10] IT168. 力攻云计算 IBM 扩大全球“蓝云”计划. http://server.it168.com/a2009/0216/265/000000265480.shtml

[11] 维基百科．Cloud computing．http://en.wikipedia.org/wiki/Cloud_computing

[12] 维基百科．云计算．http://zh.wikipedia.org/wiki/云计算

[13] Jonathan Strickland．How Cloud Computing Works．http://communication.howstuffworks.com/cloud-computing1.htm

[14] 互动百科．云计算．http://www.hudong.com/wiki/云计算

第 22 章　快速开发工具

RAD 用来描述在 60～90 天内设计和开发应用程序的方法。James Martin 于 20 世纪 80 年代中期在 Dupont工作时提出了这个概念。RAD 的基本思想是通过一组来自用户的基本需求，开发人员通常可以在工作室环境下快速地构建一个原型，用户可以与这个原型交互并建议特性、功能增强等，而这个原型则作为联合需求设计（Joint Requirements Planning，JRP）或联合应用程序开发（Joint Application Development，JAD）过程的一部分。所以 RAD 开发工具（或开发平台）应该能够方便地从 JRP 中收集用户需求，快速地创建供用户查看和修改的应用程序原型。

22.1　快速开发工具概述

构件随 RAD 概念而出现，Delphi 提供了一种方便的实现构件的方式，开发人员利用这种方式创造了大量的良好可复用构件。Borland 的构件化设计同样造就了 JavaBean，也延伸到了.Net。简单地说，RAD 是让开发人员去创造和使用可复用的构件。在基于构件的开发模式下，一般开发工作与设计工作已经分离，由不同的专职人员完成。

RAD 方法主要用于信息系统应用软件的开发，它包含如下几个开发阶段。

（1）业务建模：业务活动中的信息流被模型化，以回答如下问题，即什么信息驱动业务流程？生成什么信息？谁生成该信息？该信息流往何处？谁处理它？

（2）数据建模：业务建模阶段定义的一部分信息流被细化，形成一组支持该业务所需的数据对象。标识出每个对象的特征（称为属性），并定义这些对象间的关系。

（3）处理建模：数据建模阶段定义的数据对象变换成为要完成一个业务功能所需的信息流。创建处理描述以便增加、修改、删除或获取某个数据对象。

（4）应用生成：RAD 假设使用第四代技术。RAD 过程不是采用传统的第三代程序设计语言来创建软件，而是复用已有的程序构件（如果可能的话）或是创建可复用的构件（如果需要的话）。在所有情况下，均使用自动化工具辅助软件构建。

（5）测试及反复：因为 RAD 过程强调复用，许多程序构件已经是测试过的，这减少了测试时间。

因此，准确地说，快速开发工具应该是 RAD 工具。一个好的 RAD 工具应为研发人员提供使用基于构件的架构来快速添加和删除特性的工具，而添加和删除特性要在不需要大量重新编写代码的情况下完成。

除了在 RAD 项目的过程中更改用户需求之外，多数项目是有时间限制的，即为项

目完成设置了一个时间期限。在这个时间期限内有所有不能交付的特性或功能都应该被删除或推迟到将来的发布中。RAD 工具应支持团队中的不同角色和使用第三方构件来交付用户需求。在使用严格意义上的软件研发周期（Software Development Life Cycle，SDLC）创建应用程序的一般阶段中，在这个周期中有很多可交付物（deliverable）必须交付，包括正式的访谈、周详的设计文件和流程文件等。RAD 工具应能够快速创建解决即将到来的业务问题的应用程序。虽然有一些 SDLC 要素包括在 RAD 项目中，但这不是首要问题。例如，对于一个真正的 RAD 工具，生成程流图或数据库方案的能力并不像交付业务过程所需的功能那么重要。

22.2 常见的快速开发工具

本节简单介绍常见的一些快速开发工具，按照工具厂商分类讲解。

22.2.1 Microsoft 工具

Visual Studio 是 Microsoft 公司的开发工具套件系列产品。Visual Studio 是一个基本完整的开发工具集，它包括整个软件生命周期中所需要的大部分工具，如 UML 工具、代码管控工具、集成开发环境等。而 Visual Studio .NET 是用于快速生成企业级 Web 应用程序和高性能桌面应用程序的工具。Visual Studio 包含基于构件的开发工具（如 Visual C#、Visual J#、Visual Basic 和 Visual C++），以及许多用于简化基于小组的解决方案的设计、开发和部署的其他技术。目前，最新的版本是 Visual Studio 2008。

Visual Studio 2008 是面向 Windows Vista、Office 2007、Web 2.0 的下一代开发工具，代号为 Orcas，是对 Visual Studio 2005 一次及时、全面的升级。Visual Studio 2008 引入了 250 多个新特性，整合了对象、关系型数据、XML 的访问方式，语言更加简洁。使用 Visual Studio 2008 可以高效开发 Windows 应用。同时，Visual Studio 2008 支持项目模板、调试器和部署程序。Visual Studio 2008 可以高效开发 Web 应用，集成了 ASP .NET、AJAX 1.0，包含 ASP .NET、AJAX 的项目模板，它还可以高效开发 Office 应用和移动应用。支持.NET FrameWork 2.0/3.0/3.5。

在 Visual Studio 2008 中，对开发人员所关心的一些常见的使用场景在性能上作了很大改进，对新产品功能以及现有产品功能都设置了明确的性能指标。在 Visual Studio 2008 中显著的性能提高包括以下一些方面。

（1）重新生成一个 Visual Basic 项目并运行一个后台编译器的速度提高到了原来的 3 倍，使用的内存却只有原来的 1/3。

（2）在编辑器里滚动较大的 C#文件的速度比原来快了一倍，输入新文本的速度是原来的 1.5 倍。

（3）对 C#中庞大类型的智能感应响应时间提高了 10 倍左右。

（4）增量生成 C++/ CLI 项目解决方案的时间最快可以提高到原来的 1.9 倍。

（4）Office Word 和 Excel 文件在服务器上处理的速度快了 20 倍。

（5）TFS 版本控制命令处理经过重写后，能够在不需要绑定内存到服务器上的情况下，支持无限量的关键命令操作。

（6）充分利用多核硬件的优势来提高性能。

Visual Studio 2008 具有如下十大新功能：

（1）代码中插入图片，而且它可以被记事本程序进行编辑。它实际上是在注释中加入了另一个标签，其标签指向当前解决方案的一个图片链接。

（2）完美的代码段编辑器。在 Visual Studio 2008 中，如果想要对一个函数进行具体编码，只需要按快捷键就可以进入代码段编辑器。

（3）兼容 Linux 平台检查。它可以在多种 UNIX/Linux 下使用。而且基于这个特点，可以在 Windows 平台下编辑代码，然后可以模拟 UNIX/Linux 平台运行。

（4）即时交流功能。这个功能是在企业版才有的，而且只局限于局域网内，它需要一个服务器中心。该功能使开发人员在写代码时可以和朋友讨论问题。

（5）更漂亮的界面，真正的 3D。体现了 Windows 界面项目组的专业水平。

（6）大幅提升的 Visio 功能。不需要完全用手写代码，只需要在 Visio 中画一个一个的类、对象、关系图等。然后，选定一个生成模板，就可以基本生成一个应用程序。不过生成的代码性能不是很好，需要手工进行优化。

（7）大量专业控件。制作了上千个控件，像 ComboBox 就有多达三十多种的变种控件，并且可以上 Microsoft 的网站检索更多的控件。开发人员也可以向 Microsoft 提交自己写的控件，并且向世界共享。

（8）内嵌汇编。可以嵌入 IL 代码或本地汇编代码，不过如果嵌入了本地汇编代码，C#4.2 会提出一个编译警告。

（9）强大的配套工具。多种功能强大的工具，包括内存图形器、压力测试工具、性能分析器等，还有一个 Java2CSharp 的工具，能够转换 95%以上的代码。

（10）命令行工具。用 DOS 下的 Edit 来编写一个 Make 文件，然后写一个批处理。

22.2.2　Borland 工具

2008 年 6 月，Borland 将 CodeGear 开发工具部门卖给了 Embarcadero 公司。Embarcadero 公司原有的主要产品有 ER/Studio、DBArtisan、Rapid SQL 及 Change Manager 等。本节主要介绍原 Borland 公司的 Delphi、C++ Builder、JBuilder 和 Kylix。

1．Delphi 和 C++ Builder

Delphi 的前身是在 DOS 下的产品 Borland Turbo Pascal。Turbo Pascal 使用的是Pascal 语言。从 Turbo Pascal 5.5 版本开始，Borland 公司在传统 Pascal 的基础上加入了对象导向的功能。Delphi 是一个集成开发环境（Integrated Development Environment，IDE），使

用的核心是由传统 Pascal 语言发展而来的Object Pascal语言，通过图形用户界面作为开发环境，透过 IDE 与可视构件库（Visual Component Library，VCL）工具与编译器，配合连接数据库的功能，做成一个以对象导向设计为中心的开发工具。Delphi 程序编写后所编译的执行文件可以独立执行，容量较大些，但效率上却比较快，除了使用数据库的程序外不需安装即可执行，使用相当方便。

Borland C++是 C++集成开发环境，有 DOS 版与 Windows 版。Borland C++的前身是Turbo C++。Borland C++最后的版本是 5.02。另外，Borland C++ 5.5 仅有命令列功能。1992 年，Borland 买下 White Water 的 C++ Framework，改名为 Object Window Library（OWL），并且推出以 OWL 1.0 为核心的 Borland C/C++ 3.1，OWL 使用多重继承架构。Borland C++ 5.0 同时支持 OWL 与微软基础类（Microsoft Foundation Classes，MFC）。Borland C++ Builder 最后取代 Borland C++。

Borland Delphi 和 C++ Builder 目前的最新版本是 2009 ，即 CodeGear Rad Studio 2009。作为重要的一次版本更新，Delphi 2009 在 IDE、VCL 和语言方面都有所改进。主要有：

（1）泛型和匿名方法的编译器支持。

（2）新构件和对现有构件的增强，包括支持 Microsoft 新的 Office Ribbon 风格控件。

（3）可定制的类浏览器。

（4）项目资源管理器。

（5）增强的构建配置管理系统。

（6）升级了数据库驱动。

（7）DataSnap 更新，提供强大、灵活的处理能力，可创建不依赖于 COM 的多层应用解决方案。

（8）全面支持 Unicode，所括 IDE、语言、运行时和 VCL，及数据库访问机制。

2．JBuilder

JBuilder 是 Borland 公司开发的针对 Java的开发工具，使用 JBuilder 可以快速、有效地开发各类 Java 应用，它使用的 JDK 与 SUN 公司标准的 JDK 不同，它经过了较多的修改，以便开发人员能够像开发 Delphi 应用那样开发 Java 应用。

JBuilder 的核心有一部分采用了VCL技术，使得程序的条理非常清晰，就算是初学者，也能完整地看完整个代码。JBuilder 的另一个特点是简化了团队合作，它采用互联网工作室技术使不同地区，甚至不同国家的人联合开发一个项目成为了可能。Peloton 是基于 Eclipse 的新版 JBuilder，具有如下特点：

（1）Jbuilder 支持最新的 Java 技术，包括 Applets、JSP/Servlets、JavaBean 以及 EJB 的应用。可以自动生成基于后端数据库表的 EJB Java 类，Jbuilder 还简化了 EJB 的自动部署功能。此外，它还支持 CORBA，相应的向导程序有助于用户全面地管理 IDL 和控制远程对象。

（2）Jbuilder 支持各种应用服务器。Jbuilder 与 Inprise Application Server 紧密集成，同时支持 WebLogic Server，支持 EJB 1.1 和 EJB 2.0，可以快速开发 J2EE 的电子商务应用。Jbuilder 能用 Servlet 和 JSP 开发和调试动态 Web 应用。

（3）利用 Jbuilder 可创建纯 Java2 应用。由于 Jbuilder 是用纯 Java 语言编写的，其代码不含任何专属代码和标记，它支持最新的 Java 标准。

（4）Jbuilder 拥有专业化的图形调试界面，支持远程调试和多线程调试，调试器支持各种 JDK 版本，包括 J2ME/J2SE/J2EE。

JBuilder 最新版本为 JBuilder 2008，JBuilder 2008 不但有来自 Eclipse 开放源代码平台优点，还增强了企业级 Java IDE 的可靠性、功能性和专业性。在 JBuilder 2008 中引入了 Application Factories，重新开发了 Java IDE，包括强大的团队开发和协作特征，完整的 UML 建模性能，以及强大的代码覆盖和性能分析工具。JBuilder 2008 提供了全套可信赖的解决方案、开源下载、插件工具，框架的管理更为简便。

3．Kylix

Kylix 是 Borland 公司推出的 GNU/Linux 版的开发环境，相对于 Windows 下的 Delphi 以及 C++ Builder。通过 Kylix，程序员可以在 GNU/Linux 下使用对象 Pascal、C++或 C 语言进行软件开发。

Kylix 支持的数据库有 Oracle、Informix、DB2、Borland InterBase、PostgreSQL、MySQL。Kylix 是运行在 Linux 平台上的 RAD 开发工具。它将具有 Web 服务能力的快速电子商务开发工具引入 Linux 操作系统。Kylix 2 实际上是 Linux 下的 Dephi 版本，Kylix 3 则进一步整合了高性能的 C++ Builder 和 Delphi 程序语言，具有最前沿的开发环境、集成的调试器、直观的可视化的设计界面、综合的构件套装、Web 服务开发平台，是 Linux 平台上快速、简单地开发具有 Web 服务功能的电子商务应用程序的工具。

Kylix 的 Object Pascal 具有众多先进特性：面向对象、数组边界检查、多线程、内嵌汇编等。该编译器将直接生成优化的本机代码，而并非字节码。可以直接通过不同的编译选项对代码进行符合自己需要的优化，并可通过集成的调试器对程序进行调试和除错。Kylix 内含的编译器同时支持 ANSI C/C++（包括 Borland 扩展部分），使用过 Borland C++和 C++ Builder 的开发人员都可以立即利用 Kylix 进行开发工作。

Kylix 采用 win32 开发者熟悉的拖放方式来设计所见即所得的用户界面。它极大地提高 Linux 平台上 GUI 应用程序（尤其是商业和企业的数据库应用）的开发效率，很多 win32 商业应用将可能被移植到 Linux 平台上。

22.2.3　SUN 工具

本节简单介绍 SUN 公司的 JDK、Java Workshop 和 Java Studio 等开发工具。

1．JDK

从初学者的角度来看，采用 JDK 开发 Java 程序能够很快理解程序中各部分代码之

间的关系，有利于理解 Java 面向对象的设计思想。JDK 的另一个显著特点是随着 Java 版本的升级而升级。但它的缺点也是非常明显的，就是从事大规模企业级 Java 应用开发非常困难，不能进行复杂的 Java 软件开发，也不利于团体协同开发。

2．Java Workshop

SUN 公司于 1996 年 3 月 26 日推出了 Java Workshop 1.0，这是业界出现的第一个供 Internet 使用的多平台开发工具，它可以满足各公司开发 Internet 和 Intranet 应用软件的需要。Java Workshop 完全用 Java 语言编写，是当今市场上销售的第一个完全的 Java 开发环境，目前 Java Workshop 的最新版本是 3.0，其特点表现如下。

（1）结构易于创建：在创建平台中立的网格结构方面，Java Workshop 比其他任何一种 Java 开发工具都要方便。

（2）可视化编程：Java Workshop 的可视化编程特性是很基本的。Java Workshop 允许程序员重新安排这些操作，甚至可以确定触发操作行为的过滤器。Java Workshop 产生的模板带有许多注释，这对程序员是很有帮助的。

（3）Java Workshop 支持 JDK1.1.3 以及 JavaBeans 构件模型，API 和语言特征增加了编译 Java 应用程序的灵活性。Java Workshop 开发环境由于完全用 Java 写成，所以可移植性极好。目前，Java Workshop 支持 Solaris、Windows、UNIX 等平台。

Java Workshop 的缺点是，每一个可视化对象都迟早会用到网格布局，这种设计方法是许多人不习惯的；Java Workshop 的调色板是较差的，仅仅能满足绝大部分应用的基本要求。

3．NetBeans 与 Java Studio 5

NetBeans 是开放源码的 Java IDE，适用于各种客户机和 Web 应用。Java Studio 是一个商用全功能的 Java IDE，支持 Solaris、Linux 和 Windows 平台，适于创建和部署两层 Java Web 应用和多层 J2EE 应用的企业开发人员使用。

NetBeans 是业界第一款支持创新型 Java 开发的开放源码 IDE。开发人员可以利用业界强大的开发工具来构建桌面、Web 或移动应用。同时，通过 NetBeans 和开放的 API 的模块化结构，第三方能够非常轻松地扩展或集成 NetBeans 平台。

NetBeans 主要针对一般 Java 软件的开发者，而 Java Studio 则主要针对企业做网络服务等应用的开发者。NetBeans 与其他开发工具相比，最大区别在于不仅能够开发各种台式机上的应用，而且可以用来开发网络服务方面的应用，可以开发基于 J2ME 的移动设备上的应用等。Java Studio 为用户提供了一个更加先进的企业编程环境。

22.2.4 IBM 工具

本节主要介绍 IBM 公司的 Eclipse、Visual Age for Java、WebSphere 和 Rational 系列工具。

1．Eclipse

Eclipse 最初是由 IBM 公司开发的替代商业软件 Visual Age for Java 的下一代 IDE 开发环境，2001 年 11 月贡献给开源小区，现在它由非营利软件供货商联盟 Eclipse 基金会（Eclipse Foundation）管理。2003 年，Eclipse 3.0 选择 OSGi 服务平台规范为运行时架构。2007 年 6 月，稳定版 3.3 发布。2008 年 6 月发布代号为 Ganymede 的 3.4 版。Eclipse 目前的最新版本为 3.5。

Eclipse 是一种可扩展的开放源代码 IDE。IDE 经常将其应用范围限定在“开发、构建和调试”的周期之中。为了帮助 IDE 克服目前的局限性，业界厂商合作创建了 Eclipse 平台。Eclipse 允许在同一个 IDE 中集成来自不同供应商的工具，并实现了工具之间的互操作性，从而显著改变了项目工作流程。

Eclipse 的基础是富客户机平台（Rich Client Platform，RCP）。RCP 包括下列构件：核心平台（启动 Eclipse，运行外挂程序）、OSGi（标准集束框架）、SWT（可移植构件工具包）、JFace（文件缓冲、文本处理、文本编辑器）、Eclipse 工作台（即 Workbench，包含视图、编辑器、视角和向导）。

Eclipse 采用的技术是 IBM 公司开发的 SWT，这是一种基于 Java 的窗口构件，类似 Java 本身提供的 AWT 和 Swing 窗口构件。不过，BM 声称 SWT 比其他 Java 窗口构件更有效率。Eclipse 的用户界面还使用了 GUI 中间层 JFace，从而简化了基于 SWT 的应用程序的构建。

Eclipse 的设计思想是：一切皆外挂程序。Eclipse 核心很小，其他所有功能都以外挂程序的形式附加于 Eclipse 核心之上。Eclipse 基本核心包括图形 API、Java 开发环境外挂程序，外挂程序开发环境等。在富客户机平台上，Eclipse 使用外挂程序来提供所有的附加功能，如支持 Java 以外的其他语言。已有的分离的外挂程序已经能够支持 C/C++、Perl、Ruby、Python 和数据库开发。外挂程序架构能够支持将任意的扩展加入到现有环境中（如配置管理等），而绝不仅仅限于支持各种程序语言。

Eclipse 框架的这种灵活性来源于其扩展点。它们是在 XML 中定义的已知接口，并充当插件的耦合点。扩展点的范围包括从用在常规表述过滤器中的简单字符串，到一个 Java 类的描述。任何 Eclipse 插件定义的扩展点都能够被其他插件使用，反之，任何 Eclipse 插件也可以遵从其他插件定义的扩展点。除了解由扩展点定义的接口外，插件不知道它们通过扩展点提供的服务将如何被使用。

利用 Eclipse，可以将高级设计与低级开发工具结合在一起。Eclipse 的最大特点是它能接受由 Java 开发者自己编写的开放源代码插件，这类似于 Microsoft 公司的 Visual Studio 和 SUN 公司的 NetBeans 平台。Eclipse 为工具开发商提供了更好的灵活性，使他们能更好地控制自己的软件技术。

2．Visual Age for Java

Visual Age for Java 是一个非常成熟的开发工具，它的特性对于 IT 开发者和业余的

Java 编程人员来说都是非常用有用的。它提供对可视化编程的广泛支持，支持 EJB 的开发应用，支持与 Websphere 的集成开发，方便的 bean 创建和良好的 RAD 支持、无文件式的文件处理。

IBM 为建设 Web 站点所推出的 WebSphere Studio Advanced Edition 及其包含的 Visual Age for Java Professional Edition 软件已全面转向以 Java 为中心，这样，Java 开发人员对 WebSphere 全套工具的感觉或许会好了许多。Studio 所提供的工具有 Web 站点管理、快速开发 JDBC 页向导程序、HTML 编辑器和 HTML 语法检查等。Studio 和 Visual Age for Java 集成度很高，菜单中提供了在两种软件包之间快速移动代码的选项。这就让使用 Studio 的 Web 页面设计人员和使用 Visual Age for Java 的 Java 程序员可以相互交换文件、协同工作。

Visual Age for Java 支持团队开发，内置的代码库可以自动地根据用户做出改动而修改程序代码，这样，就可以很方便地将目前代码和早期版本做出比较。与 Visual Age for Java 紧密结合的 Websphere Studio 本身并不提供源代码和版本管理的支持，它只是包含了一个内置文件锁定系统，当编辑项目的时候，可以防止其他人对这些文件的错误修改，软件还支持诸如 Microsoft Visual SourceSafe 这样的第三方源代码控制系统。

Visual Age for Java 完全面向对象的程序设计思想，使得开发程序非常快速、高效。以致不编写任何代码就可以设计出一个典型的应用程序框架。Visual Age for Java 作为 IBM 电子商务解决方案其中产品之一，可以无缝地与其他 IBM 产品，如 WebSphere、DB2 融合， 迅速完成从设计、开发到部署应用的整个过程。

3．WebSphere

WebSphere 是 IBM 的集成软件平台，它包含编写、运行和监视全天候的 Web 应用程序和跨平台、跨产品解决方案所需要的整个中间件基础设施（如服务器、服务和工具等），提供了可靠、灵活和健壮的集成软件。

WebSphere Application Server 是该基础设施的基础，其他所有产品都在它之上运行。WebSphere Process Server 基于 WebSphere Application Server 和 WebSphere Enterprise Service Bus，为 SOA 的模块化应用程序提供了基础，并支持应用业务规则，以驱动支持业务流程的应用程序。在高性能环境中，还可以使用 WebSphere Extended Deployment 作为其基本基础设施的一部分。

WebSphere 是一个模块化的平台，基于业界支持的开放标准。可以使用受信任和持久的接口，将现有资产插入 WebSphere，并且可以随着需要的增长继续扩展环境。WebSphere 可以在许多平台上运行。

4．Rational 系列

Rational 软件是 IBM 的五大软件品牌之一，IBM 于 2003 年完成对 Rational Software 的收购。通过提高企业的软件开发能力，IBM Rational Software 可以帮助各组织机构创造商业价值。Rational 软件开发平台集成了软件工程的最佳经验、工具和服务。利用

Rational 软件开发平台，各组织机构可以获得更快的反应能力和更强的适应性，并可以集中精力关注核心任务，而代取得更大的发展。Rational 基于标准的跨平台解决方案有助于软件开发团队创建和扩展业务应用程序、嵌入式系统及软件产品。

Rational 组合提供了一个非常大的基于角色的工具集，帮助开发团队获取、创建、测试和组装服务资产，定义需求、编写业务流程代码和强制实施标准，标准化、自动化和集成业务流程及底层基础结构，使业务模型和流程与战略目标保持一致。Rational 产品包含以下 5 个类别：

（1）需求和分析，包括 Rational RequisitePro、Rational Software Modeler 和 Rational Data Architect。

（2）设计和构造，包括 Rational Application Developer、Rational Software Architect 和 Rational Systems Developer。

（3）软件质量，包括 Rational Robot、Rational Test RealTime 和 Rational Functional Tester。

（4）软件配置管理，包括 Rational ClearCase、Rational ClearQuest 和 Rational BuildForge。

（5）流程和项目管理，包括 Rational Portfolio Manager、Rational Method Composer 和 Rational Team Unified Platform。

Rational 软件的最新版本为 7.0。最新发布的 IBM Rational 软件交付平台 7.0 版本桌面产品是一套全面基于 Eclipse 的软件产品和最佳实践，有助于客户进行软件治理和强化系统开发流程，密切 IT 与业务部门之间的联系。IBM Rational 软件交付平台 7.0 版本桌面产品使客户能够进行随意选择，确保全球架构的完整性和统一性，帮助分布异地的全球开发团队更出色地设计、部署和管理软件及系统架构，确保生命周期质量。

22.2.5　Sybase 工具

本节简单介绍 Sybase 公司的 PowerDesigner、WorkSpace 和 PowerBuilder 等开发工具。

1. PowerDesigner

PowerDesigner 是一个“一站式”的企业级建模及设计解决方案，它能帮助企业快速高效地进行企业应用系统构建及再工程。IT 专业人员可以利用它来有效开发各种解决方案，从定义业务需求到分析和设计，以至集成所有现代 RDBMS 和 Java、.NET、PowerBuilder 和 Web Services 的开发等。

PowerDesigner 具有如下特点。

（1）需求管理：PowerDesigner 可以把需求定义转化成任意数量的分析及设计模型，并记录需求及所有分析及设计模型的改动历史，保持对它们的跟踪。

（2）文档生成：PowerDesigner 提供了 Wizard 向导，协助建立多模型的 RTF（Rich Text Format，富文本格式）和 HTML 格式的文档报表。项目团队中非建模成员同样可以了解模型信息，增强整个团队的沟通。

（3）影响度分析：PowerDesigner 模型之间采用了独特的链接与同步技术进行全面集成，支持企业级或项目级的全面影响度分析。从业务过程模型、UM 模型到数据模型都支持该技术，大大提高了整个组织的应变能力。

（4）数据映射：PowerDesigner 提供了拖放方式的可视化映射工具，方便、快速及准确地记录数据依赖关系。在任何数据和数据模型、数据与 UM 模型，以及数据与 XML 模型之间建立支持影响度分析的完整的映射定义，生成持久化代码以及数据仓库 ETL 文件。

（5）开放性支持：PowerDesigner 支持所有主流开发平台，支持超过 60 种（版本）RDBMS，包括最新的 Oracle、IBM、Microsoft、Sybase、NCR Teradata、MySQL 等；支持各种主流应用程序开发平台，包括 Java J2EE、Microsoft .NET、Web Services 和 Power Builder 等；支持所有主流应用服务器和流程执行语言，如 ebXML 和 BPEL4WS 等。

（6）可自定义：PowerDesigner 支持从用户界面到建模行为和代码生成的客户化定制。支持用于模型驱动开发的自定义转换，包括对 UML 配置文件的高级支持、可自定义菜单和工具栏、通过脚本语言实现自动模型转化、通过 COM API 和 DDL 实现访问功能以及通过模板和脚本代码生成器生成代码。

（7）企业知识库：PowerDesigner 的企业知识库是存储在关系数据库中的完全集成的设计时知识库，具有高度的可扩展性，便于远程用户使用。该知识库提供以下功能：基于角色的模型和子模型访问控制，版本控制和配置管理、模型与版本的变更报告以及全面的知识库搜索功能。PowerDesigner 的知识库还可以存储和管理任何文档，包括 Microsoft Office 和 Project 文件、图像和其他类型的文档。

2. WorkSpace

WorkSpace 为 Sybase ASE、IQ、ASA、EAServer、RS、UA 和 UO 等众多 Sybase 服务器提供了统一的设计和开发环境。WorkSpace 的数据管理功能覆盖了所有必要的特性：数据库建模、数据库导航、SQL 开发、SQL 调优、服务生成与消费以及调试和测试。

WorkSpace 集成了 PowerDesigner 的全部功能，同时还为 ASE 开发者提供了方便的图形化 SQL Debugger 工具，在进行存储过程调试时，可方便地对相关表进行查看和编辑。其提供的 SQL Editor 可支持语法检验、用户自定义模板、执行 SQL 语句并查看保存结果等。同时还对查询提供了图形化的查询计划，帮助开发人员分析和性能调优。

3. PowerBuilder

PowerBuilder 11.0 是目前市场上广受欢迎的 4GL RAD 开发工具，它发布了更加强大的，更加灵活的，更能提高效率的开发平台。PowerBuilder 11.0 的新技术可以帮助开发人员快速、简单地建立传统的两层应用、分布式应用、Web 应用以及智能客户端。有了

新的版本，用户可以轻松使用 DataWindow .NET 将现有 PowerBuilder 应用部署到.NET。

DataWindow .NET 是一个用于增强.NET 应用程序开发环境性能的构件，它基于几项拥有专利的强大技术，可以帮助用户快速构建和部署数据驱动的应用程序，轻松地与用户复杂的业务规程集成，发挥其在数据处理上的卓越性能。

DataWindow .NET 具备几百种内置的函数、属性和公开的程序，开发者可以在虚拟代码免费工具中体验高水平的生产力。针对 ASP .NET 的新的 WebForms 以及 Microsoft 手写识别支持，使得在应用程序中添加复杂的表格和签名识别容易实现。它具有如下优势：

（1）降低成本。大量简化了企业级数据驱动应用程序的开发和部署工作。您可以比以前更快地完成项目。

（2）提高了生产力。只需极少的编码工作，开发者在几小时或几天内就可轻松构建数据密集型应用程序。

（3）风险最小化。业已证明的技术，经过成百上千的开发人员的试用和测试，让新的开发者从成熟和强大的 4GL 图形编程构件中受益。

（4）快速的开发。内置函数和属性的 4GL 构件减少了编码工作。

（5）灵活性。利用 Datawindow 在 .NET 中的代码提高 Visual Studio .NET、Borland、C# Builder 和#develop 的效率。

22.2.6　Oracle 工具

Oracle 数据库应用程序开发主要包括建模、开发、测试、部署和监视等阶段。Oracle Application Express 和 Oracle SQL Developer 则是整个应用程序开发中最重要的两款产品。

1．SQL Developer

Oracle SQL Developer 是一个免费的图形化数据库开发工具。使用 SQL Developer，开发人员可以浏览数据库对象、运行 SQL 语句和 SQL 脚本，并且还可以编辑和调试 PL/SQL 语句。开发人员还可以运行所提供的任何数量的报表，以及创建和保存自己的报表。SQL Developer 可以提高工作效率并简化数据库开发任务，可以连接到任何 9.2.0.1 版和更高版本的 Oracle 数据库，并且可以在 Windows、Linux 和 Mac OS 上运行。

SQL Developer 包括移植工作台，它是一个重新开发并集成的工具，扩展了原有 Oracle 移植工作台的功能和可用性。通过与 SQL Developer 紧密集成，使用户在一个地方就可以浏览第三方数据库中的数据库对象和数据，以及将这些数据移植到 Oracle 数据库中。

2．Application Express

Oracle Application Express 是一个用于 Oracle 数据库的快速 Web 应用程序开发工具。仅使用 Web 浏览器以及有限的编程经验，数据库开发人员就可以开发和部署具有快速、

安全的专业应用程序。Application Express 结合了个人数据库的质量，企业数据库的生产效率、易用性和灵活性，以及 Web 的安全性、集成性、可伸缩性和可用性。开发、部署或运行 Application Express 应用程序无需客户端软件。Application Express 提供了如下三大工具。

（1）应用程序构建器：创建动态数据库驱动的 Web 应用程序。

（2）SQL Workshop：浏览数据库对象，运行即席 SQL 查询以及图形查询构建器。

（3）实用程序：允许从纯文本和电子表格上传和下载数据。

3．Oracle 的其他工具

SQL*Plus 是 SQL 命令行和 PL/SQL 语言的界面和连接 Oracle 数据库客户和服务器的报告工具。SQL*Plus 能够交替使用并通过脚本运行。

Workflow Builder 11i 是一个业务程序管理系统，支持业务员程序定义、业务程序自动化以及业务程序整合。Workflow 涉及到电子商务套件，能够建模、自动控制，以及根据自定义的业务惯例不断改善业务过程。

XML Publisher 属于 Oracle 融合中间件的基于 Java 的产品。它利用一系列的桌面工具，（如 Adobe Acrobat 和 Microsoft Word），允许用户在 XML 数据上创建自己的报表格式。

Oracle JDeveloper 10g 是一种 J2EE 开发环境，包括支持开发、调试和配置电子商务应用程序以及 Web 服务，包括一个小型的 PL/SQL 编辑器和一个 Business Process Extension Language 编辑器。

Oracle Forms Develope 是用来建立数据中心 Internet 应用程序基于 PL/SQL 的一种环境，而 Oracle Reports Developer 是给用户通过机构访问信息的一种报告工具。这两种工具都是 Oracle Developer 套件的一部分。

本章参考文献

[1] Microsoft. http://msdn.microsoft.com/zh-cn/vstudio/products/default.aspx

[2] Borland. http://www.gjj.cc/BC/Delphi/borland_mcn.htm

[3] 佚名. JDK. http://wiki.ccw.com.cn/JDK

[4] Java Workshop .http://www.xooob.com/346022_1003578.html

[5] 佚名. NetBeans IDE 特性. http://zh-cn.netbeans.org/features/index.html

[6] Sybase. http://www.sybase.com.cn/gvswse/site/china/products_solutions/develop-mentintegration. jsp

[7] Oracle. http://www.oracle.com/lang/cn/tools/index.html

第23章　多 核 技 术

多核也叫多微处理器核，是将两个或更多的独立处理器封装在一起的方案，通常在一个集成电路中。多核处理器是单枚芯片（也称为“硅核”），能够直接插入单一的处理器插槽中，但操作系统会利用所有相关的资源，将它的每个执行内核作为分立的逻辑处理器。通过在两个执行内核之间划分任务，多核处理器可在特定的时钟周期内执行更多任务。

多核架构能够使目前的软件更出色地运行，并创建一个促进未来的软件编写更趋完善的架构。操作系统专为充分利用多个处理器而设计，且无需修改就可运行。为了充分利用多核技术，应用开发人员需要在程序设计中融入更多思路，但设计流程与目前对称多处理系统的设计流程相同，并且现有的单线程应用也将继续运行。

多核技术能够使服务器并行处理任务，在以前需要使用多个处理器，多核系统更易于扩充，并且能够在更纤巧的外形中融入更强大的处理性能，这种外形所用的功耗更低，计算功耗产生的热量更少。

23.1　多核与多线程

在多核技术中，计算机可以同时执行多个进程，而在现代操作系统中，多个线程也可以并发执行。从表面上看起来，好像没有什么区别，但事实上，它们的区别是很大的。

1．多核技术与并发多线程

多核技术可以看成是一种 CPU 的集成技术，在一个 CPU 处理模块上，可以集成 2 个或者是多个 CPU，但是，它们还是单独的物理 CPU。

并发多线程（Simultaneous Multi-Threading，SMT）技术利用处理器的超标量特性，以便同时执行多条指令。SMT 技术需要操作系统的支持，是在操作系统级别上实现一个物理 CPU 的多线程并发处理，提高 OLTP 环境模式下的 CPU 利用率。它的基本理念是：没有一个单一应用可使超标量处理器达到完全饱和的状态，因此，部署同时提供输入的多个应用的效果更理想。

在程序执行时的再细分、再切割的小型化单位上，先是有进程，之后才有线程，线程的单位比进程更小，一个进程内可以有多个线程，在一个进程下的各线程，都是共享同一个进程所建立的内存寻址资源及内存管理机制，包括执行权限、内存空间、堆栈位置等，除此之外，各个线程自身仅拥有少量因为执行所必需的变量属性，其余都依据与遵守进程所设立的规定。

2．多核技术与超线程

传统的应用是单核应用，它们按顺序依次处理每一条命令。例如，用于运行三个报告的一个单线程应用会首先运行其中一个报告，当该报告完成后才开始运行第二个报告，而后是第三个。现在，许多商业应用和操作系统采用了多线程技术，可在同一时间内利用一个以上的处理器能力。它们可同时处理多个任务。在上述例子中，所有三个报告在提供多线程能力的系统上可同时进行处理，能显著的提高系统的性能。

超线程是多线程处理的一种形式，采用于大多数英特尔处理器上。超线程技术通过添加与物理线程并行的虚拟线程实现单核处理器对两个线程的执行。虽然超线程技术可提高处理性能，但潜在的提高远比不上在服务器上添加两个物理处理器核心，这是因为在处理虚拟线程时会出现超负荷处理现象。

显然，将多核技术和超线程技术进行结合，可为提供杰出的系统性能和扩展性。

23.2 多核架构

当前，多核微处理器领域除 X86 架构外，还涌现了不少最新技术成果以及面向未来的新颖概念，包括 P.A.Semi 公司的 PWRficient、SUN 公司的 UltraSPARC T1、Intel 公司的 Many Core 和 AMD 公司的 HyperTransport。

1．X86

就双核来说，AMD 和 Intel 的双核技术在物理结构上有很大的不同。AMD 将两个内核做在一个 Die（内核）上，通过直连架构连接起来，集成度更高。Intel 则是采用两个独立的内核封装在一起。因此，有人将 Intel 的方案称为“双芯”，认为 AMD 的方案才是真正的“双核”。

另外，AMD 双核处理器的两个内核并不需要通过外部通信，因为其双核处理器内部整合了一个系统请求队列（System Request Queue，SRQ）仲裁装备，每个核心将其请求放在 SRQ 中，当获得资源之后请求将会被送往相应的执行核心，所有的过程都在 CPU 核心范围之内完成，因此，AMD 双核心强调的是真正将两个核心整合在一个硅晶内核上，可以真正发挥双核心效率；而 Intel 的双核架构会遇到多个内核争用总线资源的瓶颈问题。

2．PWRficient

P.A.Semi 公司的 PWRficient 处理器采用高度整合的设计，并兼具低功耗和高性能两大特性，PWRficient 弹性的架构可被方便扩展到八核心或用于超级计算机系统，在高端服务器领域颇具竞争力。PWRficien 的指令系统的为 IBM 的 Power 设计，但 PWRficient 与 IBM 的 Power 芯片并没有太多的共同点，它拥有一套极富弹性的架构，整合度高且低功耗。PWRficient 主要定位在刀片服务器和低运营成本的服务器集群。

PWRficient 与通常的处理器逻辑非常不同，除了 CPU 内核和二级缓存外，它还包括

一个名为 ENVIO 的智能型 I/O 子系统。即 PWRficient 上包含 CPU 和 ENVIO I/O 子系统两大逻辑，两者通过一个名为 CONEXIUM Interchage 的高速交换总线联结为一个有机系统。CPU 部分为两个代号为 PA6T 的 64 位 Power CPU 内核，运行频率为 2GHz。

与其他双核芯片不同的是，PWRficient 的每一个 PA6T 内核都拥有自己的 DDR2 内存控制器，但两者是以相互独立的模式而非组成共享的双通道。此设计的好处在于每个 CPU 内核都能拥有属于自己的内存资源，最大限度降低内存抢占的概率。每个 CPU 内核都可支持 64 位或 32 位模式运作，具有诸如超标量、乱序执行、三发射等技术特性。另外，PA6T 内核也都直接整合了硬件级的虚拟技术支持，可以在多套系统同时运行时仍保持出色的性能。

3．UltraSPARC T1

SUN 公司的 UltraSPA RC T1 的重心放在多任务并行功能，这是由 UltraSPARC T1 自身的定位所决定的。UltraSPARC T1 主要针对承担网络中枢的高端服务器系统，这类服务器主要面向高吞吐量的事务计算，需要在同时处理大量的并发任务，而这些任务又都不需要复杂的运算。

SUN 公司采用非均衡的思想来设计 UltraSPARC T1，每个基本的 CPU 内核都相当精简，但都能够很好地完成相应的数据处理任务，由于精简核心占据的晶体管资源较少，处理器就能够集成更多的硬件内核；同时在较单纯的数据处理任务中，每个 CPU 核心的执行管线都不会被充分利用，在此基础上导入多线程技术将能够进一步提高系统的并行能力。我们可以看到，UltraSPARC T1 拥有 8 个对等的硬件内核，每个内核可同步执行 4 个线程，这样仅仅一枚处理器就具备同时执行 32 个不同任务的能力。

UltraSPARC T1 的晶体管总量只有 3 亿个左右，峰值能耗只有区区 80 瓦，执行效率相当出众。UltraSPARC T1 虽然具有超凡的事务处理能力，可它的科学计算能力十分糟糕，原因就在于 SUN 根据自身特殊的需要，采用不对等的设计。UltraSPARC T1 的 CPU 核心设计得非常简单，它的流水线很短，也没有包含浮点运算单元，只在 8 个核心之外附加了一个浮点运算器。这样，每个核心的晶体管占用就很少，为芯片低功耗奠定基础；UltraSPARC T1 的每个核心均只运行在 1.2GHz 的低频率下，这也是拜短流水线设计所赐，芯片节能就不难理解了。UltraSPARC T1 的每个核心都拥有 16KB 一级指令缓存和 8KB 的一级数据缓存，并具备奇偶检查能力，可以自行侦测缓存错误。

4．IBM Cell

IBM 为索尼 PS3 游戏机定制的 Cell 是一枚拥有 9 个硬件核心的多核处理器，它的多核结构同以往的多核心产品完全不同。在 Cell 芯片中，只有一个核心拥有完整的功能，被称为主处理器，其余 8 个核心都是专门用于浮点运算的协处理器。其中，主处理器只是 PowerPC 970 的精简版本，其主要职能就是负责任务的分配，实际的浮点运算工作都是由协处理器来完成。

由于 Cell 中的协处理器只负责浮点运算任务，所需的运算规则非常简单，对应的电

路逻辑同样如此，只要 CPU 运行频率足够高，Cell 就能够获得惊人的浮点效能。而由于电路逻辑简单，主处理器和协处理器都可以轻松工作在很高的频率上：Cell 起步频率即达到 4GHz 就是最好的证明。在高效率的专用核心和高频率的帮助下，Cell 接近超级计算机的水准，远远超越目前所有的 X86 和 RISC 处理器。

5. Many Core

Intel 的 Many Core 采用的也是类似 Cell 的专用化结构，Many Core 将成为 Intet 未来的 X86 处理器架构。Intet 的四核心处理器采用对等设计，每个内核地位相同，而到 Many Core 架构之后，其中的某一个或几个内核可以被置换为若干数量的数字信号处理（Digital Signal Processing，DSP）逻辑，保留下来的 X86 核心执行所有的通用任务以及对特殊任务的分派，DSP 则用于某些特殊任务的处理。

依照应用不同，这些 DSP 类型可以是 Java 解释器、动态图像专家组（Moving Pictures Experts Group，MPEG）视频引擎、存储控制器、物理处理器等等。在处理这类任务时，DSP 的效能远优于通用的 X86 核心，功耗也低得多。

如果处理器将高负载的专用任务转交给 DSP 执行之后，那么主核心的运算压力就大大减轻，系统整体效能将获得明显提升。

6. HyperTransport

AMD 公司的 Many Core 是利用现有的 HyperTransport 连接架构，对现有的 Opteron 多路服务器系统进行拓展。现有的 Opteron 多路系统并非采用共享前端总线的方式连接，而是借助专用的 HyperTransport 总线实现芯片间的直连。每一颗 Opteron 处理器都可以直接与其他的处理器进行数据交换或缓存同步，不必占用内存空间，无论系统中有多少数量的 Opteron，整套系统都能够保持高效率的运作。在该套平台中，HyperTransport 总线处于中枢地位，而它除了作为处理器连接总线外，还可以连接 PCI-X 控制器、PCI Express 控制器以及 I/O 控制芯片，也就是充当芯片间的高速连接通路。

AMD 的一套协处理器扩展方案也是以此为基础，即为多路 Opteron 平台开发各种功能的协处理器，这些协处理器都通过 HyperTransport 总线与 Opteron 处理器直接连接。对 Cray 提出的需求，AMD 给出的解决方案就是，将八路 Opteron 中的一颗 Opteron 处理器置换成矢量协处理器，以此实现矢量计算性能的大幅度增长，而 Opteron 平台本身不需要作任何形式的变动。

23.3 多核编程

多核处理器带来了强大的计算能力，如果无法实现软件程序的并行，那么，将面临大量计算资源被闲置。毫无疑问，这是一次全新的挑战。面对挑战，一批新的技术纷纷崛起。并行计算语言平台（例如，Erlang 等）、并行库（例如，Open MP 等）、并发编程（如 Java Concurrency 等）、多核计算工具（如 Intel C++ Compiler 等）成为计算 2.0 的基

础软件架构。而这些新的技术和处理器共同形成了新计算环境的底层平台，起到了地基的作用。

多核编程技术主要包括并行计算、共享资源分布式计算、任务分解与调度、Lock-Free 编程等内容。其中共享资源分布式计算、任务分解与调度是最重要的内容，也是大多数程序员未接触过的内容，许多并行算法都可以通过它们来实现。多核编程模式主要是提供一种多核并行与分布式编程的普遍方法，有了这些编程模式后，程序员不再需要去学习各种复杂的并行算法，它可以复用现有的串行算法，轻易地实现并行和分布式计算。

希赛教育专家提示：在多核编程技术中，最重要的是如何将计算均匀分摊到各个 CPU 核上。

多核时代的到来，给程序员的编程思维带来了巨大的冲击。为了能够充分利用多核性能，程序员必须学会以分块的思维设计程序、以多进程或多线程的形式来编写程序。到底应该使用多进程还是多线程的形式来编写程序，是最让程序员感到困惑的问题之一，笔者认为，需要根据具体的应用来决定。但是，在通常情况下，使用多线程进行多核编程比使用多进程有更大的优势。

（1）线程的创建和切换开销比进程更小。

（2）线程间通信的方式多而且简单也更有效率。

（3）多线程有汗牛充栋的基础库支持。

（4）多线程的程序比多进程的程序更容易理解和修改。

除了编程形式，使用多线程编程的动机也发生了改变。在以往，对于 Windows 程序员来说，使用多线程的主要原因之一是为了提高用户体验，如在长时间的计算中提高 UI、I/O 或者网络的响应速度。而在多核时代编写应用程序为了充分利用多个计算核心，缩短计算时间或者在相同的时间段内计算更多任务。例如，游戏编程时通过多线程的方式把碰撞检测的计算分散到多个 CPU 内核可以大大缩减计算时间，也可以利用多核做更细致的检测计算，从而能够模拟更加真实的碰撞。

在多核时代，对编程语言的选择也要更加谨慎。无论开发何种项目，相对于 C/C++/Fortran 等编译型语言，C#/java/Python 等脚本语言也许是更好的选择。原因在于脚本语言比较高级，一般都提供了对多线程的原生支持。相形之下，编译型语言往往都是通过平台相关的库来提供多线程支持的。由于没有统一的标准，造成使用 C/C++编写多线程程序时需要考虑更多的细节，提高了项目成本。各种语言对多线程支持的比较如表 23-1 所示。

表 23-1　各种语言对多线程支持的比较

语言 / 支持	C/C++等编译型语言	C#/java/Python 等脚本	PHP/Ruby/Lua 等脚本
语言支持多线程	否	是	否
库支持多线程	是	是	否

续表

语言 支持	C/C++等编译型语言	C#/java/Python 等脚本	PHP/Ruby/Lua 等脚本
支持内核级线程	是	是	否
支持用户级线程	可模拟	可模拟	是
线程编程复杂度	一般/易	易	N/A

虽然 C/C++在多线程编程方面因为没有从语言级提供支持而失去了部分优势，但因为当前的主流操作系统都以 C 语言接口的方式提供创建线程的 API，而 C/C++又有相当丰富的程序库，也就一定程度上弥补了语言上的不足。使用 C/C++编写多线程程序不仅可以使用 Win32 SDK，还可以使用 POSIX threads、MFC 和 boost.thread 等。虽然这些库都提供了一定程度的封装，减轻了程序员进行多线程负担，但对于目标定位于提升计算密集型程序的性能的多核程序员来说，这些方式仍然太为复杂。因为使用这些库几乎要增加一倍的关键代码，相应地调试和测试的成本也大大增加。

更好的选择应该是使用 OpenMP 这种通过编译器加强来支持多线程的基础库。OpenMP 通过使用#pragma 编译器指令来指定并行代码段，对程序的改动相当少。而且，可以指定编译为串行版本以方便调试，更可以和不支持 OpenMP 的编译器共存。

希赛教育专家提示：虽然脚本语言在语言层次上提供了对多线程编程的原生支持，但却并没有比 C/C++领先多远，根本原因在于脚本语言的基础（数据结构与算法的基础库）与 C/C++基础库一样，是以串行形式来设计开发的。

本章参考文献

[1] 李宝峰，富弘毅，李韬. 多核程序设计技术——通过软件多线程提升性能. 北京：电子工业出版社，2007

[2] 佚名. 将软件推向未来多核架构. http://www.diybl.com/course/3_program/gcs/200836/102895.html

[3] 倪天福. 多核编程和编译技术. http://news.chinabyte.com/306/2434306.shtml

[4] 佚名. 多核编程的趋势，程序员的新历程. http://blog.csdn.net/gavin_luo/archive/2008/12/24/3593640.aspx

[5] 佚名.多核与多线程技术的区别. http://server.zdnet.com.cn/server/2008/0723/1005241.shtml

第 24 章　片 上 系 统

片上系统（System on Chip，SoC）指的是在单个芯片上集成一个完整的系统，对所有或部分必要的电子电路进行包分组的技术。所谓完整的系统，一般包括中央处理器、存储器，以及外围电路等。

目前，很多具有 CPU 功能的主流消费性电子产品（如视频转换器、移动电话和 PDA 等）中的芯片，都可称之为 SoC 芯片。这类产品对成本与市场价格相当敏感，因此，产品的竞争力便来自于谁能更好地控制成本。由于在一个芯片中集成了多种不同的功能模块，SoC 技术为降低消费类电子产品的成本提供了机会。更重要的是，SoC 具有应用领域的行为和功能特征，具有更多的应用专业知识，使整机成本、体积以及功耗都大大降低，加快了整机系统更新换代的速度。

24.1　SoC 的组成与优点

一个典型的 SoC 会由以下各元件组成：

（1）一个或多个微控制器、微处理器或 DSP。

（2）内存部分会包含只读内存、随机存取内存，电可擦可编程只读存储器（Electrically Erasable Programmable Read-Only Memory，EEPROM）和快闪存储器。

（3）振荡器，提供系统所需时脉来源。

（4）其他周边设备，包括计数器、实时时钟芯片（Real-Time Clock，RTC）或 Watch-dog 等。

（5）额外的界面，包括通用串行总线（Universal Serial BUS，USB）、FireWire、Ethernet、通用同步/异步串行接收/发送器（Universal Synchronous/Asynchronous Receiver/Transmitter，USART）等。

（6）类比界面，包括类比-数位转换器及数位-类比转换器。

（7）电压调整电路及电源管理。

这些设备会用工业标准总线，让数据从设备直接传送到动态内存，而不需要经过处理器，这样，可以增加 SoC 处理数据的速度。

SoC 与单功能芯片相比，有如下优点：

（1）增加功能，从单一功能增加到多种功能。

（2）提高性能指标，在相同的工艺条件下可实现更高性能的系统指标，同时，采用 SoC 设计方法完成同样功能所需的晶体管数目可降低 2～3 个数量级。

（3）减少体积，降低所占的印制电路板（Printed Circuit Board，PCB）空间。一个芯片集成一个系统，相当于一个部件或一部整机，势必减少整机的体积。

（4）缩短产品上市时间，以获取更多的利润。如数字视频光盘（Digital Video Disc，DVD）播放器用 SoC 后的研制时间可从原来的 1～2 年缩短到 6 个月以内。

（5）降低功耗，提高抗电磁干扰和系统可靠性。例如，日本日立公司的 HG73M 系列 SoC 集成了 SH3 型内核、高速逻辑电路和高密度动态随机存储器（Dynamic Random Access Memory，DRAM）等，其数据传输速率比采用外部 DRAM 系统高 10～100 倍，其功耗仅为原来的 10%～50%。

（6）降低成本。SoC 要集成多种功能的集成电路（Integrated Circuit，IC），要缩短设计周期，必须向别的公司购买知识产权（Intellectual Property，IP），可减少重复劳动、提高效率、节约开支、降低成本，并且减少了外围电路芯片，也降低了整机的成本。

（7）缩短供需双方的距离，SoC 制造商要更多地与用户联系，尽早让用户参与设计，这样设计出来的芯片上市最快、最受用户欢迎，也最容易占领市场。

SoC 为实现许多复杂的信号处理和信息加工提供了新的思路和方法。SoC 采用了片内可再编程技术，可使片上系统内硬件的功能可以像软件一样通过编程来配置，从而可以实时地进行灵活而方便的修改和开发。这种全新的系统设计概念，使新一代的 SoC 具有极强的灵活性和适应性。它不仅使电子系统的设计和开发以及产品性能的改进和扩充变得十分简易和方便，而且使电子系统具有适应多功能的能力。

24.2 SoC 与 SiP

随着半导体工艺技术的发展，IC 设计者能够将愈来愈复杂的功能集成到单硅片上。SoC 正是在 IC 向集成系统转变的大方向下产生的。从狭义角度讲，它是信息系统的芯片集成，是将系统集成在一块芯片上；从广义角度讲，SoC 就是一个微小型系统，如果说 CPU 是大脑，那么 SoC 就是包括大脑、心脏、眼睛和手的系统。SoC 的出现使 IC 发展成为集成系统，整个电子整机的功能将可以集成到一块芯片中。在不久的将来，集成电路与电子整机之间的界限将被彻底打破。

SoC 就是将微处理器、模拟 IP 核、数字 IP 核和存储器（或片外存储控制接口）集成在单一芯片上。它通常是客户定制的，或是面向特定用途的标准产品。

SoC 是面向特定用户的，能最大满足嵌入式系统要求的芯片，因而具有很多优势。例如，能极大改善功耗开销，可减少印制板上部件数和管脚数，减少板卡失效的可能性，有利于板卡的性能改善（由于片内连线缩短），降低风冷要求，减少系统开发商成本，尤其适合数字化产品开发，如手持设备、信息家电等。

集成电路特征尺寸的缩小，使得 SoC 似乎成为必然的发展方向；然而，对于同时拥有多种材质、多种工艺的系统，系统级封装（System in Package，SiP）是最好的选择。

集成方式的选择应充分考虑芯片加工工艺、产品性能及设计周期的要求。

用尽可能小的空间、尽可能低的功耗实现产品功能和性能的优化，是集成电路从业者追求的目标，SoC 和 SiP 是达到这一目标的两条不同途径。随着电子整机向多功能、高性能、小型化、便携化、高速度、低功耗和高可靠方向发展，10 多年来，SoC 和 SiP 都有快速的发展。

SoC 研发的复杂性对知识的要求越来越高，专用集成电路（Application Specific Integrated Circuit，ASIC）的设计者可以只掌握某一个特定领域的知识，而 SoC 则要求设计者对系统有全面的了解，才能开发出适用性广的产品。复杂性的增强同时也导致了研发周期的延长，研发的成本也越来越高。

SiP 在一定程度上则可以弥补这一缺点。SiP 在一个封装中组合多种 IC 芯片和多种电子元器件，以实现与 SoC 同等的功能。20 世纪 90 年代后期，美国佐治亚理工学院 RaoR.Tummala 教授提出了一种典型的 SiP 结构——单级集成模块（Single Level Integrated Module，SLIM）。SLIM 将各类 IC 芯片、光电器件、无源元件、布线和介质层都组装在一个封装系统内，极大地提高了封装密度和封装效率，其封装效率可达 80%以上。相比之下，目前国际流行的球栅阵列（Ball Grid Array，BGA）封装工艺的封装效率也仅为 20%。

SiP 技术在无线通信领域的应用颇受业界青睐。由于商用射频（Radio Frequency，RF）芯片很难用硅平面工艺实现，使得 SoC 技术能实现的集成度相对较低，性能难以满足要求。同时，由于物理层电路工作频率高，各种匹配与滤波网络含有大量无源器件，SiP 的技术优势就在这些方面充分显示出来。SiP 在封装架构方面允许高度的灵活性，尤其对 RF 应用具有很大的优势；同时，它还允许更低的功耗和更低的噪音，在混合与匹配 IC 工艺方面具有灵活性。

至于 SoC 和 SiP 究竟孰优孰劣，显然不能一概而论。在产品数量大、性能高、集成度高的应用中，SoC 自然当仁不让；如果要求高适应性、面市时间短、多种不同功能集成在一个封装内的应用中，则 SiP 占有优势。

24.3　SoC 设计

SoC 将电子系统几乎全部的功能集成到一块芯片上，从而在单个芯片上能实现数据的采集、转换、存储、处理和 I/O 等多种功能。换一个角度来说，SoC 能集成数字电路、模拟电路、硬件专用电路、存储器、微处理器、DSP 等多种异构模块，实现多个复杂的应用功能。

系统复杂性源于两个因素，一个因素是集成度的增加使得单一芯片上晶体管数目成指数级增长，另一个因素是消费者对电子产品更多功能，更低价格的需求，以及更短上市时间的压力。根据 ITRS1999，单一芯片上晶体管数目平均每年增长 58%，而设计工

程师每人月平均能完成设计的晶体管数目平均每年只增长 21%，随着时间的推移，设计能力与制造能力之间的差距将越来越大。

24.3.1 设计概述

一般而言，SoC 系统都具备强大的数据处理和存储能力，能适应于一类典型应用的功能和性能要求。同时，它还应当具有灵活的软硬件可编程能力。软件的可编程性体现在基于同一种类型的架构下，充分发挥软件本身的适应性和可重用性，减小更改硬件所带来的开销，提高设计速度。硬件的可编程性在于通过对应用的特殊功能定制满足性能要求的专用加速器，充分利用硬件本身的便携性和可扩展性，减小研发软件而带来的额外费用，缩短设计周期。因此，SoC 的设计应该是一个软件、硬件协同设计的过程，这也是 SoC 系统一个非常重要的标志。

然而，传统的集成电路设计方法一般都是将系统分为两个阶段：系统级软件开发部分和电路级硬件设计部分。特别指出的是，软件开发和硬件设计往往是相对独立进行的。在系统级，软件开发人员使用诸如 C/C++等高级编程语言进行系统描述和算法仿真，并分析系统在软件层面的各项指标，撰写系统设计书，然后移交给硬件设计工程师。在电路级，硬件设计师首先要花大量的时间理解系统设计书，之后才能利用高速集成电路硬件描述语言（Very-High-Speed Integrated Circuit Hardware Description Language，VHDL）或 Verilog 硬件描述语言进行电路设计。在此手工转换的过程之中，可能还会引入人为的错误因素。另外，为了验证软件开发的正确性，必须等到硬件全部完成之后才能开始进行软件测试和系统集成，大大延长了设计的进程。

传统的设计方法使得在软件和硬件之间很难进行早期的平衡和优化，并有可能严重影响开发成本和开发周期。根据有关统计，从系统级设计到电路级设计所花费的时间一般是系统级设计所花时间的 3 倍左右。因此，在系统级设计与电路级设计之间架设一座桥梁，已经成为 IC 设计领域极为迫切的任务。

图 24-1 概括了面向 SoC 系统级的关键技术，包括软硬件协同设计技术、设计重用技术、与底层相结合设计技术，三者相辅相成、相互促进。

软硬件协同设计技术常与设计重用技术交织在一起，成为目前 SoC 系统级设计的主要部分。而与底层相结合的高层设计技术是在现阶段由于制造工艺不断进步，进入纳米级环境的前提下，提出的一种能有效解决高层综合和物理设计不匹配而导致设计不收敛问题的新技术。

24.3.2 软硬件协同设计技术

软硬件协同设计课题的提出已有多年的历史，但是，早期的研究多集中在针对一个特定的硬件如何进行软件开发或根据一个已有的软件实现具体的硬件结构。前者是一个经典的软件开发问题，软件性能的好坏不仅仅取决于软件开发人员的技术水平，更赖于所使用的硬件平台；后者是一个软件固化的问题，实现的途径可以是采用一个与原有软

件平台相同的软件处理器，并将软件代码存储于存储器中，也可以是在充分理解软件的内在功能之后完全用硬件来实现软件的功能。

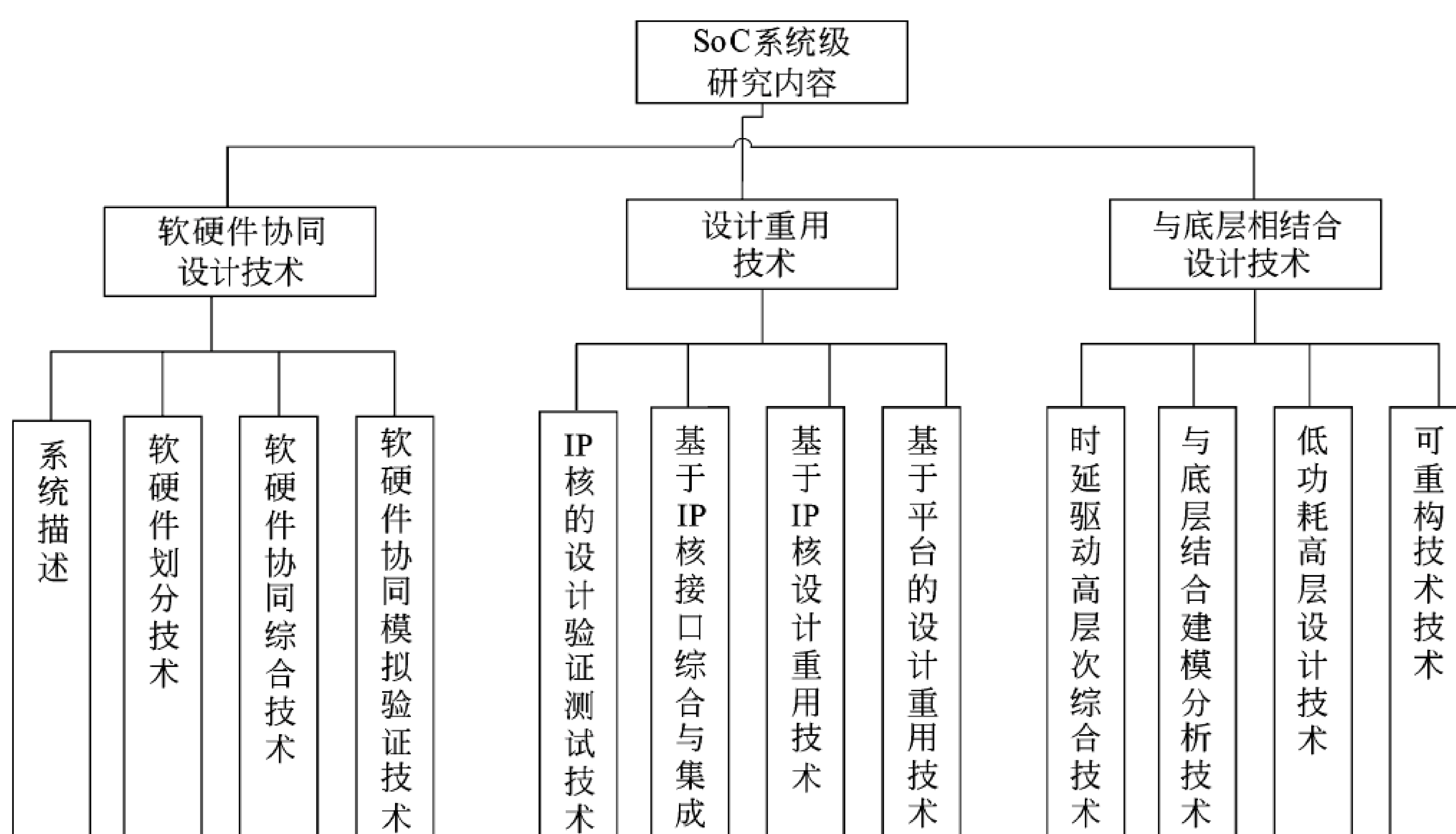


图 24-1　面向 SoC 系统级的关键技术

一般来说，采用存储器固化软件代码的做法可以比较快地实现芯片设计，且芯片具有一定的二次开发可能。但是，由于考虑到实现所需的硬件平台的一致性，芯片的性能将受到较大的限制，大多应用在性能比较低的场合。除此之外，有时候要找到一个可用的、与软件开发时所使用的硬件平台兼容的处理器也是一件十分困难的事情。将软件功能全部由硬件来实现的做法具有较大的风险，一般需要比较长的时间和比较大的人力、物力和财力的投入，特别是进入市场的时间较为苛刻时，这种做法有其局限性。但是，一旦成功，则芯片具有较高的性能。

从上述介绍不难发现，早期的软硬件协同设计方法研究还是一种面向目标的（Object Oriented）软硬件设计方法，研究的内容和结果与所要实现的目标和已具备的条件密切相关，不能形成具有普遍适用性的理论体系。

面向 SoC 的软硬件协同设计理论是从一个给定的系统任务描述着手，通过有效地分析系统任务和所需的资源，采用一系列变换方法并遵循特定的准则，自动生成符合系统功能要求的，符合实现代价约束的硬件和软件架构。这种全新的软硬件协同设计思想需要解决许多以前没有碰到的问题。

首先，是系统的描述方法。目前，广泛采用的硬件描述语言（Hardware Description Language，HDL）是否仍然有效，如何来定义一个系统级的软件功能描述或硬件功能描述等。至今，尚没有一个大家公认的，且可以使用的系统功能描述语言可供设计者使用。

其次，是这一全新的设计理论与已有的 IC 设计理论之间的接口。可以预见，这种全

新的设计理论应该是现有 IC 设计理论的完善，是建筑在现有理论之上的、一个更高层次的设计理论，它与现有理论一起组成了更为完善的理论体系。

第三，这种全新的软硬件协同设计理论将如何确定最优性原则。显然，沿用以往的最优性准则是不够的。除了芯片设计师们已经熟知的速度、面积等硬件优化指标外，与软件相关的（如代码长度、资源利用率、稳定性等）指标也必须由设计者认真地加以考虑。

第四，如何对这样的一个包含软件和硬件的系统的功能进行验证。除了验证所必须的环境之外，确认设计错误发生的地方和机理将是一个不得不面对的课题。

最后，是功耗问题。传统的 IC 在功耗的分析和估计方面已有一整套理论和方法。但是，要用这些现成的理论来分析和估计含有软件和硬件两部分的 SoC 将是远远不够的。简单地对一个硬件设计进行功耗分析是可以的，但是，由于软件运行引起的动态功耗，则只能通过软硬件的联合运行才能知道。

在 SoC 中，尤其是在面向特定应用领域的 SoC 中，软硬件的结合是非常紧密的，软件和硬件之间的功能划分，以及它们的实现并没有固定的模式，而是随应用的不同而变化。

目前，在软硬件设计中最为活跃的研究工作包括系统描述、软硬件划分、软硬件协同综合，以及软硬件协同模拟与验证。图 24-2 给出了一个较为普遍的面向 SoC 软硬件协同设计流程。

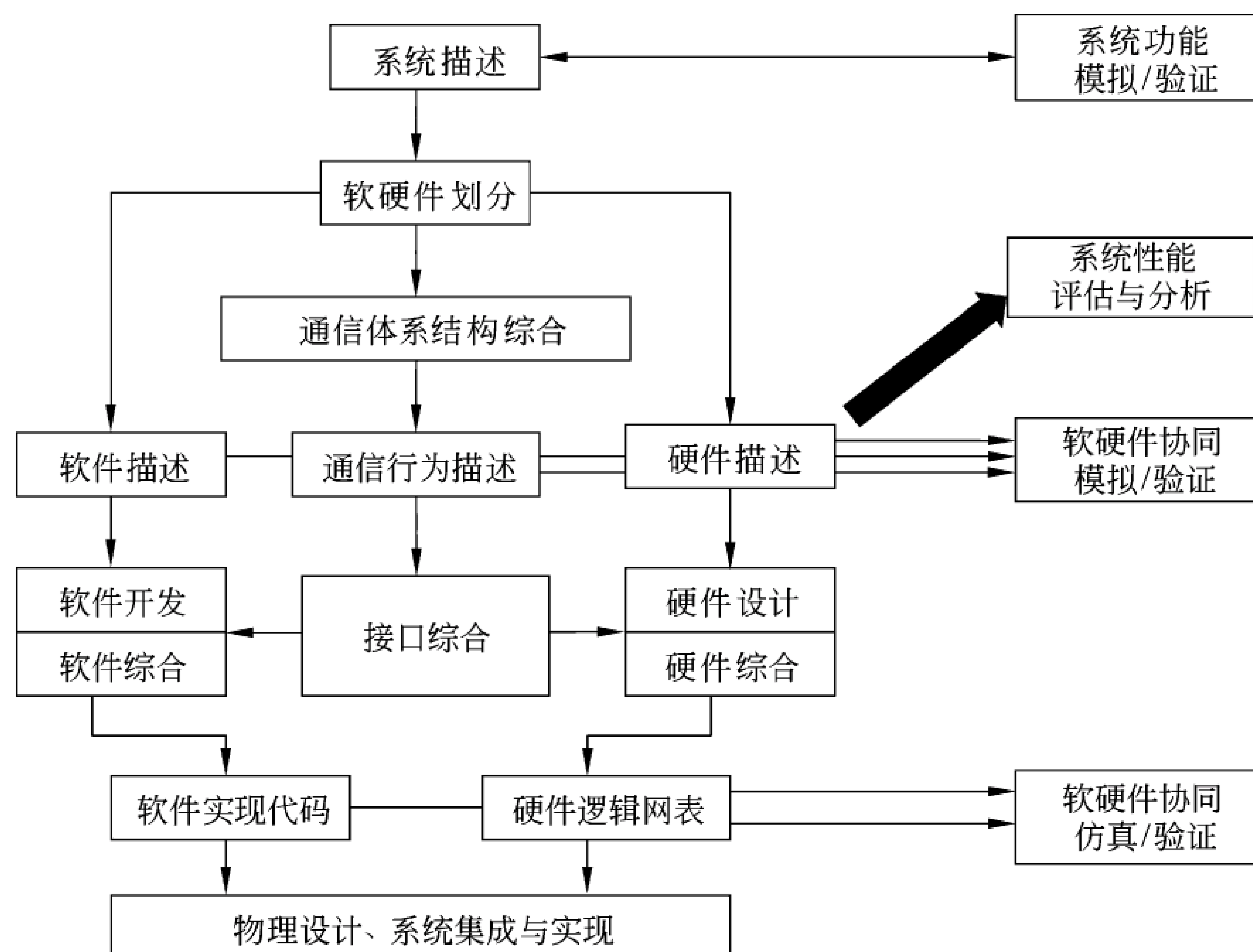


图 24-2　面向 SoC 软硬件协同设计流程

24.3.3　设计重用技术

SoC 芯片的集成度越来越高，投放市场的时间要求越来越短，为了实现满足一定功能需求和性能要求的系统，设计者越来越依赖于重用技术的支持。SoC 的设计重用技术主要可分为基于 IP 核的模块级重用和基于平台的系统级重用。

基于 IP 核的设计重用主要包括 IP 核的设计和 IP 核的使用。IP 核的设计目标是实现即插即用，但目前离这个目标还有较长的距离。IP 核的设计除了需要考虑具体功能之外，还要考虑可重用、可测性及测试的可重用性。IP 核的质量是 IP 核最重要的因素之一，IP 核必须是可重用、可配置和可升级的，而且 IP 升级应符合可重用标准以确保升级后 IP 核的可重用性。

基于平台的设计重用是近几年提出的设计重用方法，它是基于 IP 核设计重用技术的扩展，延伸了设计重用的理念，强调系统级重用。基于平台的设计方法要求提供面向特定应用领域的设计模板，设计者通过对设计模板进行适当的修改来构造符合性能要求的 SoC 系统。

在设计重用技术的推动下，基于 IP 核的接口综合与集成将逐步成为研究的热点。接口综合在不同的年代赋予了不同的意义，从 20 世纪 80 年代末不同单元块间接口逻辑电路的生成，到 90 年代中不同构件间固定转换电路的使用，到现在不同 IP 核间自适应接口电路的设计，使接口综合技术得到了相应的发展。特别是在当前以 IP 核复用技术为主的 SoC 时代，接口综合的地位越显突出，它已成为 SoC 设计中的关键技术之一。

正如前面所讨论的，今天在单个芯片上已可以集成上千万乃至上亿只晶体管。芯片变得如此复杂，它实现了以前需要许多块 PCB 甚至机架才能完成的功能。在这样高的集成度下，设计的难度已变得非常高，设计代价事实上主导了芯片的代价。这不仅要求设计者必须具备系统和芯片两方面的知识，同时也必须充分考虑市场竞争的压力，最大限度地缩短设计周期。凡事从零做起的思路显然不能适应这种新情况，而采用前人成功的设计经验和设计资料是解决这个问题的明智选择。

所谓设计重用实际上包含两个方面的内容，涉及设计资料重用技术和如何生成可被他人重用的设计资料。前者通常被称为 IP 重用，因为可以反复使用的设计资料通常具备比较复杂的功能，且经过验证。设计资料内不仅仅包含一些物理功能和技术特性，更重要的是包含了设计者的创造性思维，具有很强的知识内涵。

1. IP 核的生成

IP 核实际上就是一个经过验证的 IC 设计，从其实现的形式和应用层次上来看，IP 核可以有三种不同的表现形式，即软核（Soft-Core）、固核（Firm-Core）和硬核（Hard-Core）。

软核以 HDL 的方式提交，其性能通过时序模拟进行验证。由于软核不依赖于任何实现工艺或实现技术，具有很大的灵活性。使用者可以方便地将其映射到自己所使用的工艺上去，可复用性很高。软核的另外一个重要的优点是使用者拥有全部源代码。使用

者可以通过修改源代码，方便地生成同样功能且自有版权的新软核，从而避免向原有软核作者支付版税。同时聪明的软核使用者还可以通过增加自己的知识和经验，产生出远比原始软核广泛得多的新的软核。正是由于软核的上述优点，它的价格不菲，且提供者寥寥无几。

但是软核也有自身的弱点。由于软核的载体是硬件描述语言且与实现的工艺无关，使用者在最终将其嵌入自己的设计时就要对从描述到版图的转换的全过程负责。虽然这要涉及经曲 IC 设计的全部内容，IC 设计人员必须具备相当的风险意识。另外，工艺映射和系统的性能有着一定的内在关系，是否可以不加修改地将一个软核映射到任何一个工艺上仍然是需要探讨的一个问题。

硬核以 IC 版图的形式提交，并经过实际工艺流片验证。显然，硬核强烈地依赖于某一个特定的实现工艺，而且在具体的物理尺寸，物理形态及性能上具不可更改性。这些特点对使用者来说有喜有忧。喜的是硬核已经过验证并具有最优的面积代价和性能的设计，使用者不需考虑与上相关的优化问题。忧的是硬核与工艺的强相关性迫使使用也只能使用核工艺完成电路其他部分的设计，而且要在布局布线遵守注意固核的物理限制。

处于软核和硬核之间的固核以电路网表的形式提交并通常采用硬件进行验证。硬件验证的方式有很多种，例如，可以采用可编程器件进行验证，采用硬件仿真器进行验证等。固核往往对应于某一个特定的实现工艺，在该实现工艺的条件下固核具有最优的面积和性能特性。对于使用者来说，不需要对固核的功能给予过多地关注，可以减少许多相关的设计工作，同时由于固核的时序特性是经过严格检验的，设计者只要保证在布局布线过程中电路关键路径的分布参数不会引起时序混乱就可以保证芯片的设计成功。

但是固核也有其自身的缺点，那就是它与实现工艺的相关性及网表的难读性。与实现工艺的相关性限制了固核的使用范围，网表的难读性则使得布局布线后发生的时序违反的排除变得比较困难。由于固核在使用的方便程序上和开放程序上均介于软核和硬核之间，其价格也处于它们的价格之间。

2．IP 核与常规 IC 不同的地方

IP 核的生成具有与常规的 IC 设计不同的特点。例如，时序、测试和低功耗等虽然是 IC 设计中的经典问题，但是，直接将已有的设计方法应用到 IP 核的设计中，就会出现许多意想不到的困难。

3．IP 核面临的新问题

IP 核的使用也面临许多新问题。由于 IP 核的特殊性和 IC 开发的高风险性，IP 核的使用决不是这些 IP 核的简单堆砌，使用过程中不仅仅要考虑它们的功能，更要使它们溶入芯片。IP 核的使用需要综合考虑诸多因素，而不是几个 IP 核的简单堆砌。

24.3.4 与底层相结合设计技术

高层次综合技术（High Level Synthesis，HLS）自问世以来，受到了广泛的重视。但

随着集成电路制造工艺的不断进步，集成电路的复杂度也进一步提高。目前仍旧占据设计主流的实时逻辑（Real Time Logistics，RTL）级开始的设计方法已经面临挑战。特别是在物理寄生效应已经成为电路性能和成本的主导因素的今天，高层次综合的作用日显突出。它不仅仅需要考虑传统的调度、分配和控制器综合内容，还需要考虑物理寄生效应。这给高层次综合的形式化增加了更大的难度。

与底层结合的高层次综合技术主要包括时延驱动的高层次综合技术、与底层结合的高层建模与分析技术、高层低功耗设计技术。

可重构计算技术是一个新兴的研究领域，它是以可编程逻辑芯片为硬件基础，完成应用功能的设计。最为突出的一个特点是，它能根据应用的需求动态配置电路的实现形式，具有灵活的软件编程性，同时具备专用集成电路的高性能，因此，即使是不同的应用在同一可重构计算硬件平台上也能获得非常高的计算加速比。

以布尔代数为基本理论基础的现代数字 IC 设计技术面向的是系统的功能设计，布尔代数定义的各种基本逻辑运算所描述的是一个系统的输出对输入的逻辑关系。这种逻辑关系以一组包含 0 和 1 两个基本逻辑值的逻辑向量来表示。现代 IC 设计的核心问题就是解决如何准确地实现这种用二值逻辑确定的系统功能，或者说找到一组正确描述系统功能的逻辑表达式。

24.3.5　设计方法与流程

SoC 设计方法和传统的集成电路设计方法有很大的不同，如图 24-3 和图 24-4 所示。

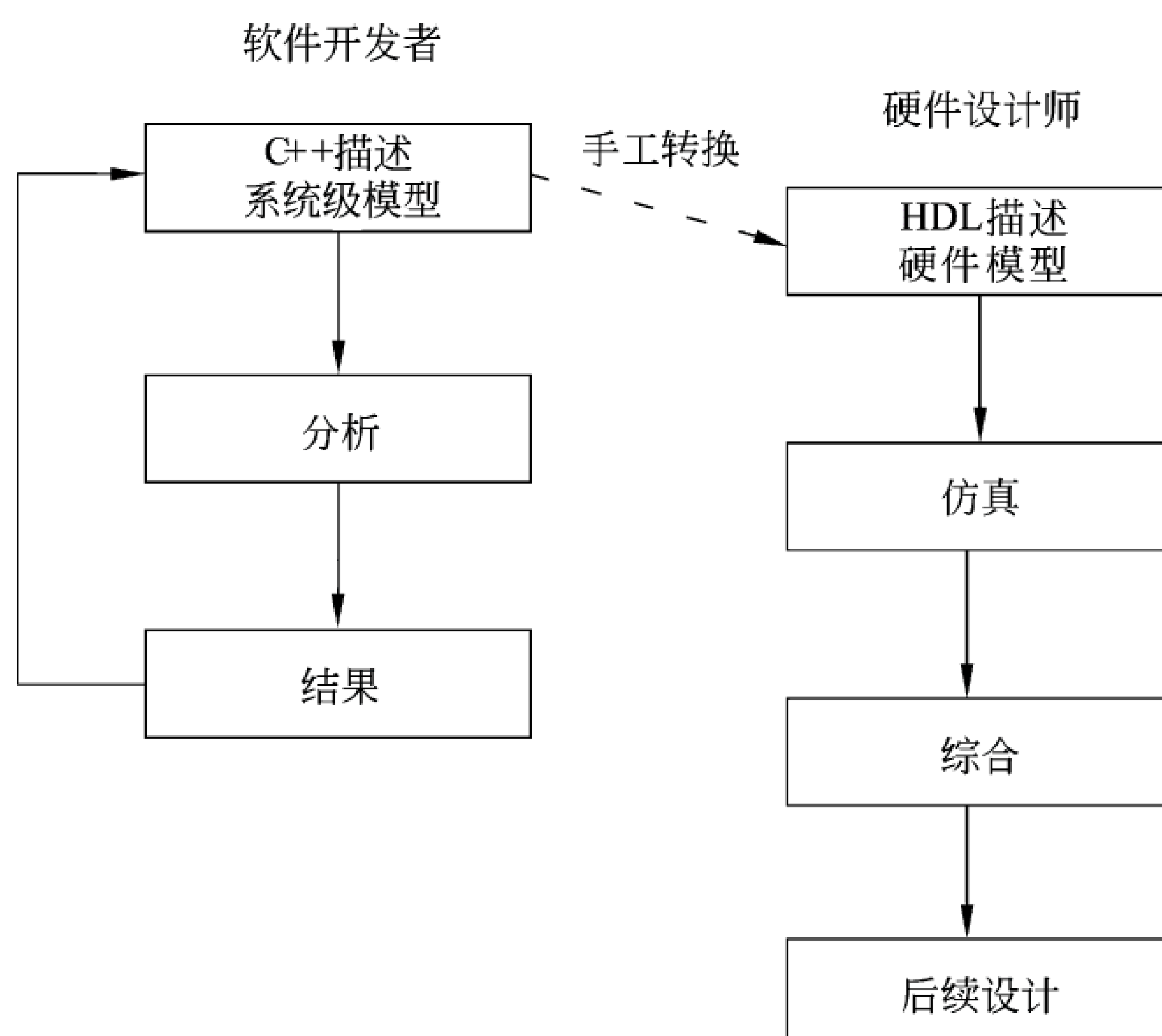


图 24-3　传统的集成电路设计方法

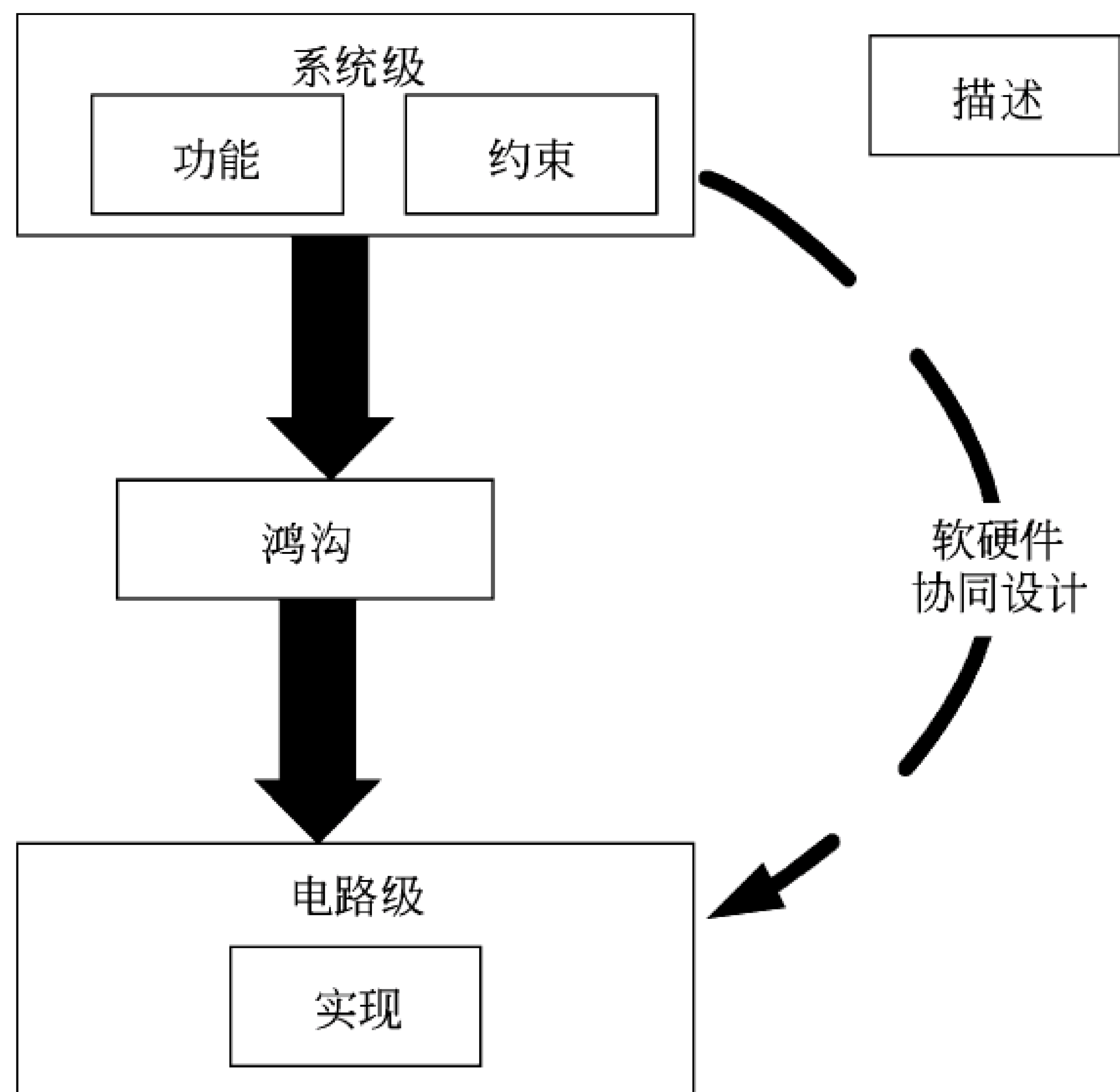


图 24-4　软硬件协同设计方法

SoC 系统级设计方法学在国外的研究十分活跃，许多厂商、研究机构以及大学都开发了适应于 SoC 系统级的设计工具，并提出相应的设计流程和设计方法。这些方法基本上是采用层次化（Hierarchical）的设计思想和正交性（Orthogonalization）的设计原则来完成系统级设计。

目前，在学术界和工业界比较认可的 SoC 系统级设计方法学，从性质上基本可分为三大阵营，分别是自顶向下、自底向上、上下结合或中间相遇。表 24-1 列举了三种不同的系统级设计方法的主要优缺点。

表 24-1　三种不同的系统级设计方法比较

系统级设计方法	优　　点	缺　　点
自顶向下、细化求精设计方法	符合软件开发者、硬件设计师的思路；易于定义层次关系，明确层次行为、结构和语义；易于开发建模/划分/综合/仿真工具	较强地依赖于某种系统级设计语言；设计重用效率降低，开发的产品只能定制于某一种应用
自底向上、搭积木设计方法	易于遵循设计重用思路；能简化设计流程，加快设计速度；扩充对体系结构探索能力	系统的集成较为困难，通信接口综合问题比较严重；较强的依赖与地层环境的支持
上下结合、分而治之设计方法	易于遵循计算与通信、行为结构相分离的设计原则；与具体的设计语言无关；开发的产品适用于一类应用，有较强的可编辑性、灵活性	平台的定义比较复杂，需兼顾软件开发与硬件设计；难于开发面向平台的自动综合与验证工具

图 24-5 是 SoC 设计流程的一个简单示意图。

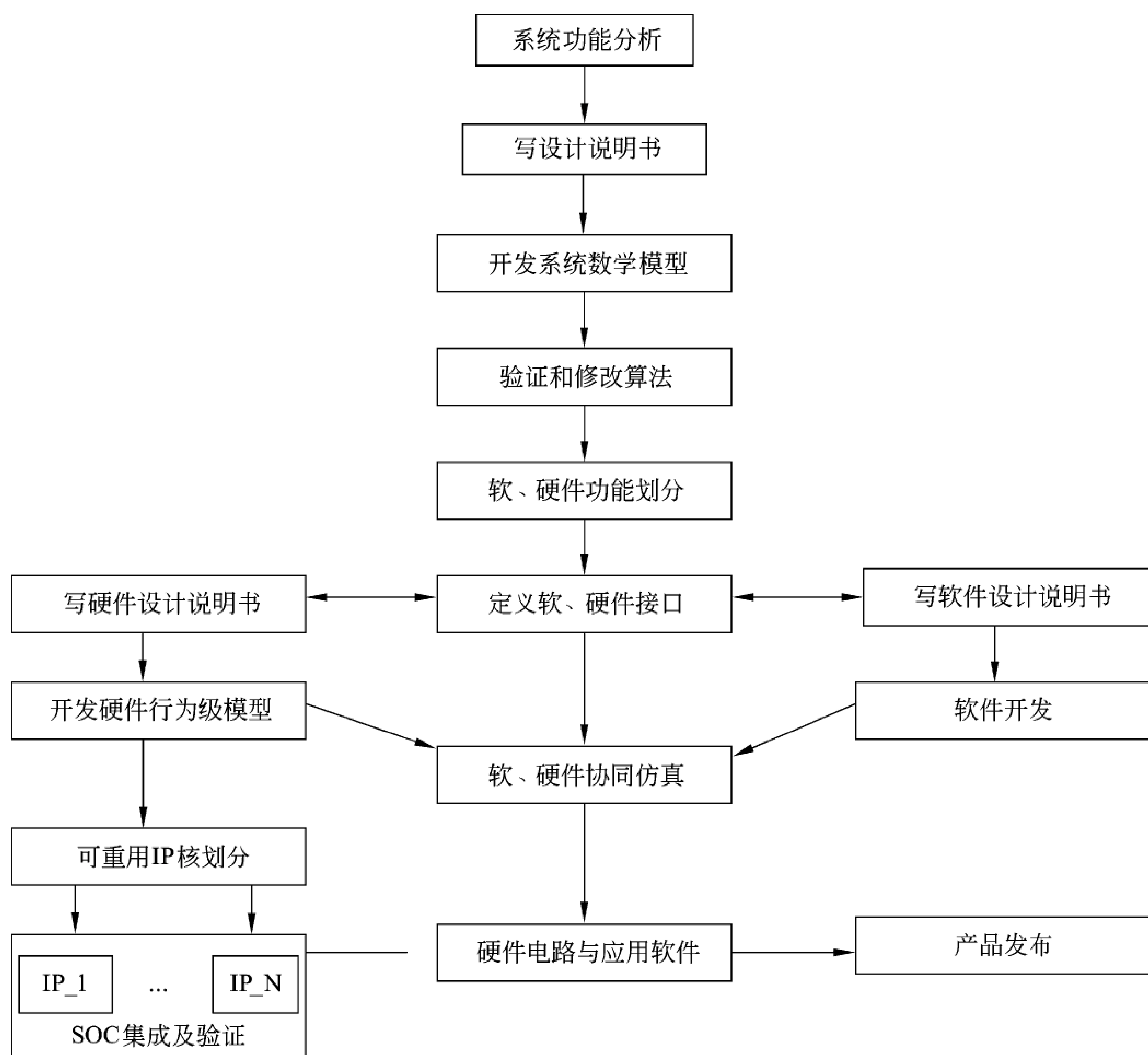


图 24-5　SoC 设计流程图

24.4　SoC 验证

Janick Bergeron 对验证的定义是：证明一个设计的功能是否正确的过程。SoC 的验证工作贯穿整个设计流程，从行为级 HDL 设计，一直到芯片设计定案之前都需要做足够多的验证工作，当前，验证工作量已经占整个设计工作量 70%左右。总体而言，验证、模拟和验证工作成为整个 SoC 学科发展的制约瓶颈，是提高设计生产率的障碍。

目前芯片一次投片成功率在 35%左右，造成重复投片的主要原因就是验证不够充分。验证是件困难的事，单一的设计工具难以解决验证问题，而需要一系列复杂的工具和技术，来减少设计错误数，使之达到可接受的程度。

SoC 验证研究内容很多，例如，IP 核/模块级验证、系统级验证、仿真验证、软硬件

协同验证、等价性检查、静态时序分析和时序验证、版图验证等。随着验证技术的逐步发展，验证方法由最初的直接测试向量生成、约束随机测试，到覆盖驱动验证和最新的基于断言的验证方法，各种验证方法在不断创新发展。

1．模块/IP 核级验证

任何 SoC 设计均由一系列模块组成。模块可能是自己开发，也可能是重用第三方的 IP 核。不论哪种情况，在系统集成前做 IP 核验证工作是必需的。

（1）软性检查：主要检查代码语法、可综合性、变量未初始化、结构化可支持性和端口失配性等。

（2）规范模型检查：主要做设计特征遗漏性检查，以在早期发现错误状况。

（3）功能验证：主要利用基准测试向量基于事件或基于时钟进行功能验证，如黑盒测试、白盒测试和灰盒测试等。

（4）协议检查：主要验证是否违反总线协议或模块互连约定，按照协议逐一检查并比较结果。

（5）直接随机测试：通过随机产生数据、地址、控制等信号检查功能正确性，减少模拟仿真工作量。

（6）代码覆盖率分析：主要根据模拟仿真时统计代码被执行数，可以按陈述句、信号拴（Toggle）、状态机、可达状态、可触态、条件分支、通路和信号等进行统计分析，以提高设计可信度。

2．系统级验证

系统级验证主要确认芯片体系结构满足所赋予的功能/性能要求。系统级设计阶段将用户需求转换成功能/性能要求，并实现行为/功能设计，然后映射到相应的体系结构上（设计输入、硬 IP 核、软 IP 核、软/硬件划分、性能分析、总体优化、性价比评估等反复迭代），最后进行系统级验证。

在系统级验证中，往往要构建虚拟目标系统。首先，做功能验证，验证是否满足要求；其次，做软硬件性能验证；第三，做系统级基准测试（自顶向下验证策略），抽取特定功能，编制测试向量/程序，定义对错条件，覆盖所有功能，形成基准测试程序（反复迭代），用于模拟仿真。

在复杂 SoC 设计开发中，模拟仿真占整个验证工程师团队工作量的 40%～70%，由于成本和市场压力，寻找灵巧的仿真技术十分迫切。

（1）功能仿真：主要关注模块之间互连验证、系统总线协调性验证和标准规范兼容性验证等，由于复杂度高，可通过事件驱动和加速技术，例如，硬件加速器、模拟发生器和快速建模试验等来加速和简化仿真工作。

（2）基准测试包：首先搭建 SoC 整体架构，然后将每一模块经基准测试包挂接到系统总线上。这些基准测试包有利于识别缺陷，但它们不是设计工作的一部分，而是为了验证而引入的。基准测试包测试向量来自于 IP 核供应商、直接随机产生、手工编制，或

由系统级测试捕获。

（3）事件驱动仿真：使用比较普遍，但受芯片规模和性能限制。首先设计代码被仿真工具所接受，其次编制基准测试向量，再运行仿真，最后通过单步调试，错误定位、改正后可再次仿真。

3．FPGA 验证

随着半导体制造技术的不断前进和相应的设计规模以及复杂度增加，使得传统的软件仿真工具已不可能完全解决功能验证的问题。而且，一些需要处理大量实时数据的应用（如视频）也越来越多，因此，要求能够在接近实时的条件下进行功能验证。

现场可编程门阵列（Field Programmable Gate Array，FPGA）验证成为 SoC 设计流程中重要的一个环节，一方面作为硬件验证工具，可以将所设计的 RTL 级代码综合实现后写入 FPGA 芯片进行调试检错；另一方面可以进行软件部分的并行开发，在验证板上检测驱动程序、启动操作系统。FPGA 验证的流程相当于一个 FPGA 设计的主要流程，它主要分为设计输入、综合、功能仿真（前仿真）、实现、时序仿真（后仿真）、配置下载、下载后板级调试检错等几个步骤。

总的来说，FPGA 验证是整个 SoC 设计中一个重要而且有效的验证步骤，用来改进 RTL 级设计代码，验证功能的正确和完整性，提高 SoC 流片成功率。

希赛教育专家提示：一个高质量的验证计划使得验证工程师可以更早地开发标准测试程序环境。这种并行的开发验证环境，能尽早给验证团队一个明确的目标，也是保证验证可重用的关键。为了得到一个高质量的验证计划，验证工程师要正确和充分地理解设计需求和规范，要与设计工程师及时地交互，这样才能保证验证计划的易读、易用和可重用。因此可以说，一个好的验证计划可以有效提高验证效率，缩短开发周期，在 SoC 开发中有着重要的意义。

本章参考文献

[1] 魏少军. 未来 SoC 技术发展的几个特点. http://www.eepw.com.cn/article/89289.htm

[2] 佚名. SoC 设计的关键技术. http://www.laogu.com/wz_58288.htm

[3] 吴武臣. 复杂 SoC 设计. 北京：机械工业出版社，2006

[4] 马光胜，冯刚. SoC 设计与 IP 核重用技术. 北京：国防工业出版社，2006

第 25 章　面向方面的编程

面向切面编程（Aspect Oriented Programming，AOP）是 OOP 的延续，它可以通过预编译方式和运行期动态代理，实现在不修改源代码的情况下给程序动态的、统一地添加功能。AOP 实际上是 GoF 设计模式的延续，设计模式孜孜不倦追求的是调用者和被调用者之间的解耦，AOP 可以说也是这种目标的一种实现。

25.1　AOP 概述

AOP 的目标是提供方法和技术，把问题分解为一系列的相关功能构件和一系列贯穿多个功能构件的方面，然后组合这些构件和方面，获得系统的实现。由于 AOP 是一种新的开发技术，其引入了很多新的概念，为了后面讨论的方便，本节先介绍 AOP 中的一些基本概念：

（1）方面（Aspect）：对横向分布在多个对象中的关注点所做的模块化。方面定义可能包括一些代码，以及在何处、何时、怎样调用这些代码的指令。基于不同的方面语言，方面可以层次化地构建，方面语言可以提供独立的机制来定义某个方面，并指定其与底层系统的交互。例如，对象交互、内存管理、永久性、历史化、安全、缓存策略、配置、监控、测试、数据的结构和表示，以及领域特定优化等。

（2）连接点（Joinpoint）：程序执行过程中的一个特定点，用来定义在程序的哪个地方通过 AOP 加入新的逻辑，例如，对某个方法的调用或某个特定异常的抛出都可以称为连接点。连接点模型提供了通用的引用框架，允许定义方面的结构。

（3）通知（Advice）：AOP 框架在某个特定的连接点处所运行的代码。通知有多种类型，包括环绕通知，前置通知和异常通知等。通知是不知觉的，因为在连接点没有明确的符号指出要在此运行通知，编写原始基础代码的程序员可能不知道这种变化。

（4）切入点（Pointcut）：指通知的应用条件，用于确定某个通知要被应用到哪些连接点上。AOP 框架允许让开发人员指定切入点。

（5）引入（Introduction）：向目标对象添加方法或字段的行为。

（6）目标对象（Target object）：指包含连接点的对象。如果一个对象的执行过程受到某个 AOP 的修改，那么它就叫做目标对象。

（7）代理（Proxy）：由 AOP 框架在将通知应用于目标对象后创建的对象。

（8）编织（Weaving）：是一种将核心功能模块与方面组合在一起，从而产生出一个工作系统的过程。编织可以在编译期完成，也可以在运行时完成。

同步、实时限制、错误检查都是方面的常见例子，其他还有很多方面的例子，例如，对象交互、内存管理、永久性、历史化、安全、缓存策略、配置、监控、测试、数据的结构和表示，以及领域特定优化等。许多方法在某种系统中一起出现。例如，分布式系统的某些方面，包括构件交互、同步、远程请求、参数转换策略、负载平衡、赋值、失效处理、服务的质量和分布式事务。

25.1.1　与 OOP 的比较

OOP 在过去的几年里已经成为主流，几乎完全取代了过程式编程。面向对象编程的一个最大的优势是整个软件系统可以被看成是由一些独立的类的集合组成，其中每个类都有良好定义的任务，它的责任是非常清晰的。在面向对象的程序中，这些类相互协作以完成整个应用程序的目标。但是，系统的有些部分不能被看成是某一个类的责任，它们跨越了整个系统。例如，日志记录、对上下文敏感的错误处理、性能优化、协调工作、内存管理、实时约束，以及其他趋向于横切多个功能构件的系统行为（将这种行为称为“横切关注点”），因为它跨越了多个模块，而当前的软件开发技术使用一维的方法学来处理这种需求，把对应需求的实现强行限制在一维的空间里。这个一维空间就是核心模块级实现，其他需求的实现被迫嵌入这个占统治地位的空间中，现有的这种将 N 维系统需求实现为一维空间的开发方法，导致了糟糕的需求到实现的映射，从而有以下表现。

（1）代码混乱：软件系统中的模块可能要同时兼顾几个方面的需求，兼顾多个方面的需求导致相应关注点在实现元素的同时出现，从而引起代码混乱。

（2）代码分散：由于横切关注点本来就涉及到多个模块，相关实现也就遍布在这些模块当中。例如，在一个使用数据库的系统中，性能问题就会影响所有访问数据库的模块，从而导致代码的分散。

AOP 可以解决这些问题，它允许开发者动态地修改静态的 OO 模型，构造出一个能够不断增长以满足新增需求的系统，就像现实世界中的对象会在其生命周期中不断改变自身，应用程序也可以在发展中拥有新的功能。AOP 利用一种称为“横切”的技术，剖解开封装对象的内部，并将那些影响了多个类的行为封装到一个称为“方面”的可重用模块。

AOP 是一种新的编程技术，其概念的出现不过几年的时间，却体现了非常简单而深刻的“分而治之”的解决问题思想。“分而治之”是指把一个复杂的问题分解成若干个简单的问题，然后逐个解决，其思想的核心是分散关注点。从软件架构角度上看，分散关注点的思想在设计之初专注于核心需求，其他各种辅助功能的需求作为核心功能需求的扩展和补充。因此，系统各个功能模块均可以看作是系统的关注点，一个复杂的系统就是由多个关注点来组合实现的。

AOP 与 OOP 最重要的区别在于它处理横切关注点的方式。在 AOP 中，每个关注点实现都不知道其他关注点是否会“关注”它。例如，信用卡处理模块不知道其他的关注点实现正在为它做日志和验证的操作。AOP 展现了一个从 OOP 转化过来的强大的开发

范型。一个 AOP 实现可以借助其他编程范型作为基础，从而原封不动地保留其基础范型的优点。例如，AOP 可以选择 OOP 作为它的基础范型，从而把 OOP 善于处理一般关注点的好处直接带过来。用这样一种实现，独立的一般关注点可以用 OOP 技术，正如过程型语言是许多 OOP 语言的基础一样。因此，AOP 也不会抛弃现有的 OOP 和结构化程序设计等技术的思想精髓。

希赛教育专家提示：AOP 构建在 OOP 系统上，可以说 OOP 是 AOP 的技术基础，AOP 是对 OOP 的继承和发展。相比于传统的程序设计方法学，AOP 能够将关注点分离，从而模块化地实现贯穿特性。当使用 AOP 编程时，可以结合需求/问题的特点，选择合适的实现结构，即过程、对象或方面。

25.1.2 软件开发过程

AOP 包括三个清晰的开发步骤，分别是方面分解、关注点实现、方面的重新组合，如图 25-1 所示。

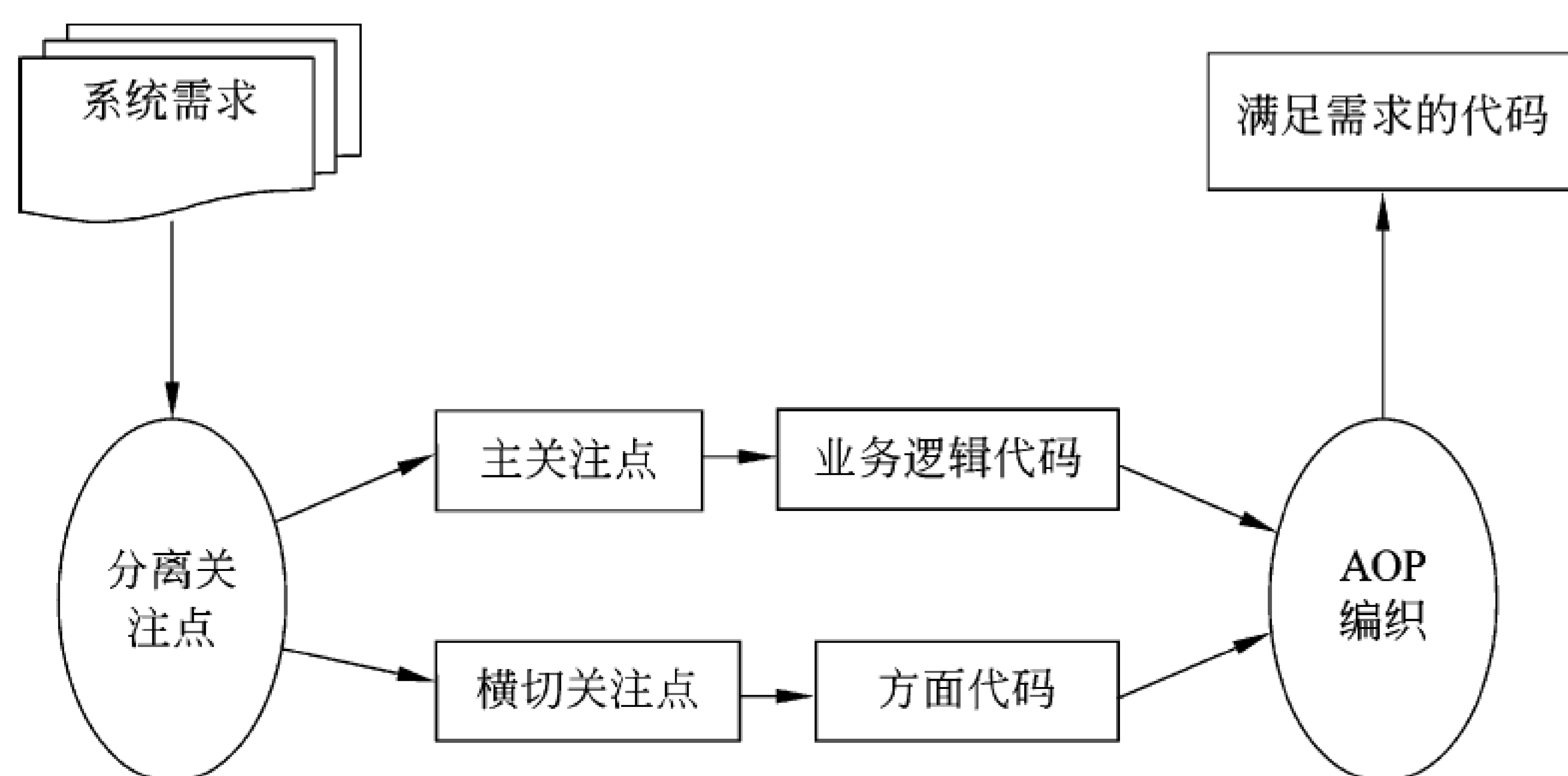


图 25-1 AOP 的开发步骤示意图

（1）方面分解。分解需求提取出横切关注点和一般关注点。把核心模块级关注点和系统级的横切关注点分离开来。例如，对于一个信用卡系统，可以分解出三个关注点，即核心的信用卡处理、日志和验证。

（2）关注点实现。各自独立地实现这些关注点，用 OOP 实现一般关注点，用 AOP 实现横切关注点。例如，可以用 OOP 实现信用卡处理单元，而用 AOP 实现日志单元和验证单元。

（3）方面的重新组合。方面集成器通过创建一个模块单元（方面）来制定重组的规则，重组过程也称为编织。

25.1.3 优点和应用领域

AOP 除了可以解决代码混乱和代码分散所带来的问题外，它还有一些别的好处：

（1）模块化横切关注点。AOP 用最小的耦合处理每个关注点，使得即使是横切关注点也是模块化的。这样的实现产生的系统，其代码的冗余小。模块化的实现还使得系统容易理解和维护。

（2）系统容易扩展。由于方面模块根本不知道横切关注点，所以，很容易通过建立新的方面加入新的功能。另外，当往系统中加入新的模块时，已有的方面自动横切进来，使系统易于扩展。

（3）设计决定的迟绑定。使用 AOP，设计师可以推迟为将来的需求作决定，因为他可以把这种需求作为独立的方面很容易地实现。

（4）更好的代码重用性。AOP 把每个方面实现为独立的模块，模块之间是松散耦合的。松散耦合的实现通常意味着更好的代码重用性，这一点上，AOP 比 OOP 做得更好。

通常，可以在如下情景中使用 AOP 技术：权限、缓存、内容传递、错误处理、懒加载（Lazyloading）、调试、记录跟踪优化、性能优化、持久化、资源池、同步、事务处理。

25.2　AOP 的相关技术

AOP 作为一种新型的编程技术，与之相关的主要技术有关注点分离、反射、横切、编织等，本节简单介绍这些技术。

25.2.1　关注点分离

随着当今软件应用复杂性的不断提高，新出现的创新技术使得程序必然要融合和处理许多特殊的计算关注点，如并发、分布式、实时约束、位置控制、持久性和失败恢复等。在所有这些专用关注点之下的是基本关注点，基本关注点主要负责基本的计算算法和基本功能，而专用关注点的存在是为了完成应用的特殊需求，或是为了管理和优化基本的计算算法。集成一个附加关注点的典型方法是扩展一个给定的程序语言，提供小部分新的编程语言构件来处理关注点。

关注点在概念上是独立的，当将它们混合在代码中就会带来许多问题：

（1）编写混合的代码很困难而且非常复杂，因为所有的关注点必须在同一级别同时得到处理。扩展的编程语言在实现级无法提供足够的关注点抽象。由于缺乏抽象，混合的代码很难理解。

（2）由于关注点是强耦合的，混合代码难以维护和修改。针对面向对象系统，由于不同关注点的强耦合，混合代码产生了继承的不规则性。不可能重定义一种方法实现，或混合不同的关注点到子类中。

对于软件关注点，可以区分两种不同级别的分离，分别是概念级和实现级。

在概念级，分离关注点需要解决两个问题：

（1）对每个关注点提供足够的抽象作为单个概念。

（2）确保单个关注点是原语级，在某种程度上它们处理的是程序员思想中的自然关注点。

在实现级，分离关注点需要提供足够的组织架构隔离出关注点。在这一级的目标是分离出一块代码，处理不同的关注点，为它们提供松散耦合。

要分离特定关注点，通常有三种方法，分别是元级编程、自适应编程和组合过滤器。

1．元级编程

元级编程是一种反射系统为表示自身而融合的结构。编程语言的基本构件（如类和对象调用）在元级进行描述，并可以用元编程进行扩展和重定义。每个对象通过一个元链接与一个元对象相关。元对象负责在基本对象上的操作语义。

元级编程支持实现级的关注点分离，通过捕获发送到对象的消息和对象接收的消息，元对象能够执行特殊目的的工作。例如，它们可以检查同步约束，确保实时声明，在不同的机器之间移植参数，写日志等。这允许编写基础级算法而不包括专用关注点，而这些反过来又可以在元对象中编程。同样，由于有了结构化反射，元级编程可以实现算法和数据组织之间的分离。

2．自适应编程

自适应编程是一种基于代码模式的编程模型。代码模式是按照不同类别划分的，每个模式捕获了编程中的抽象。

（1）传播模式。定义在数据上的操作。传播模式标识了为特殊操作而交互的类的子图。对数据的引用是通过附属子图声明，以一种结构化隐含方式实现的，实际的代码在沿传输路径的代码封装器中进行定义。

（2）传输模式。抽象了参数化过程。它们用在传播模式中，沿着子图将参数传进或带出。

（3）同步模式。在并发应用中定义对象间的同步机制。它们的目的是控制进程对操作执行的访问。

每个模式类处理了不同的关注点，可以作为高层软件构件，它们通过名字消解，以一种松散耦合的方式与其他构件交互。每种模式独立于其他模式和数据组织。这会产生一些影响，例如，在类组织中的变化不一定导致模式中的修改，对算法（传播模式）的修改不一定导致同步机制的改变。

3．组合过滤器

组合过滤器模型通过添加对象组合过滤器，对传统面向对象模型进行了扩展。在此，将重点集中在过滤器部分，以及它们怎样实现分离的。

过滤器是第一类的对象，也是过滤器类的实例。过滤器的目的是管理、影响发送和接收的消息。特别是过滤器指定了消息接受和拒绝的条件，并确定了合适的结果行为。过滤器可在每个类的基础上编程。系统确保相应的方法执行前，消息要由过滤器处理。

一旦收到一条消息，它必须通过一系列输入的过滤器；在发送消息之前，还必须通过一系列输出过滤器。

分离关注点是通过为每个关注点定义一个过滤器来实现的，每个过滤器类负责处理其相关关注点的所有方面。过滤器机制使程序员有机会捕获所有接收和发送的消息，在方法代码实际执行前执行特定的行为。结果代码因此被很好地分解成专用关注点（在过滤器中）和基本关注点（在方法中）。

以上技术的共同点是它们提供了一种机制来解释发送和接收的消息。元对象协议在元级执行了这个解释，是通过计算反射和消息具体化实现的。组合过滤器通过内置的过滤器机制捕获消息。在这两种情况下，解释器是在运行时执行的。面向模式的编程实现了编译时的消息解释，模式编译器检测何时为专用关注点所需要方法扩展代码，并将该段代码直接插入，它类似于预处理器（在运行时不做解释）。

元级编程的一个重要的方面是分离关注点不能加载到模型上。还有元级编程通过提供有关语言本身构件的反射信息，非常利于分解关注点。在元级编码专用关注点是一种策略，程序员可以采用或不采用它。这与过滤器和代码模式相反，它们提供了专用的语言构件来实现关注点的分离。结果，过滤器方法和代码模式方法需要对每种新的关注点创建新的语言构件，而在元级编程中则不是这样的情况。

25.2.2　反射技术

反射是系统观察和改变自身执行的能力。当一种编程语言提供了实体的显式表示（即具体化）时，这些实体或是程序构造块（如类、方法），或是包括在程序执行（如栈、垃圾收集器）过程中的，这种编程语言称为有反射性。因此，使用反射性语言，开发者不仅可以定义系统功能，而且可以定义新伪程序建立块或执行机制（即定义功能是怎样执行的）。换句话说，开发者不仅可以编写程序，还可以扩展解释器。

为了显示反射和 AOP 之间的关系，需要一种从 AOP 概念到这些反射的映射。这个映射解释了反射是怎样用来分离方面代码和基础代码，以及怎样把它们编织在一起的。

要把方面代码从基础代码中分离出来，要借助于基本级和元级之间的自然分离。基本代码定义在基本级，方面定义在元级。基本对象表示基本代码，元对象表示方面。分离一个个方面定义，是通过为每个方面使用一组特定的元对象集合来实现的。这样，每个元对象参与到单个方面中。

编织是用元链和元对象协作来实现的。使用元链把基本代码编织成方面。事实上，元链连接元对象和基本对象。通常，使用基本代码编织多个方面时，一个给定的基本对象需要连接到多个元对象。当两个元对象完成了控制同一个基本对象的同样的执行机制时，就会产生冲突。这种冲突可以通过元对象协作得以解决。与编织有关的一个重要的概念是联接点。联接点对应基本代码执行流中的特定点。

反射是达到 AOP 目标的一个强大工具。事实上，反射开放了编程语言，使之具有

可扩展性和可适应性。然而，反射由于其复杂性而受到指责。其基本概念难以学习，而且使程序难以理解。然而，这种复杂性至少是部分隐藏的。方面开发者可以使用一些能够编译成反射语言的专门的方面语言。专门的方面语言简化了表达方面，限制了对已有反射功能的访问。

使用 AOP 编程的一个问题在于缺少合适的开发工具。尤其是对调试而言。构建这样工具的困难在于这样一种原因：一般来说，方面往往无法独立测试，而且难以通过分析编织的代码来检索特定的方面代码。

反射提供了使用现有开发工具的好处。方面可以用元对象类来表示，可以使用类浏览器来查看和编辑元对象的代码。测试和调试也可以使用现有工具来实现。由于元对象方面之间相互独立，调试就更容易一些。实际上，每个方面由一组甚至在编织之后都彼此独立的元对象表示。可以说，这些工具相当低级，不适合方面。然而，它们是工具问题的第一个答案，它们可以扩展（特别是通过反射）来更好地支持方面开发。

软件重用是软件工程中的一个圣杯。为了在不同应用中重用方面，方面应该是泛化的，即方面的定义不应该显式地引用到应用构件。然而，编织需要连接到应用构件。这在某种程度与方面泛化是相对立的。

通过使用元对象表示方面，反射带来了 AOP 重用的解决方案，这种解决方案是由面向对象编程提供的。可以使用由现有元对象类继承来的新的元对象类，可以很容易地构建新的方面。甚至，元对象类可以容易地做成泛化的（即独立于任何应用）。事实上，只有连接元对象到基本对象的配置脚本才必须是特定于应用的。

25.2.3 编织技术

编织是实现 AOP 的一个重要机制，利用方面编织器可以把 AOP 实现的方面代码织入到用 OOP 实现的核心功能代码中，从而构建最终系统。编织的实现机制有多种，从编织的过程来看可以分为两类，分别是静态编织和动态编织；而根据编织发生在应用程序生命周期的不同时刻，又可以分为三类，分别是编译时编织、载入时编织和运行时编织。

1．静态编织和动态编织

静态编织是指在核心功能代码中的适当位置（例如，某段代码执行前或执行后），将方面代码织入，从而形成混合的编码；而动态编织可以在程序运行时，根据上下文决定调用的方面和它们的先后顺序以及增加或删除一个方面等。利用静态编织的工具有 AspectJ、AspectC++等；其他一些使用反射技术或运行时代码织入技术的体系结构都属于动态编织，这样的工具有 AspectWerkz、Jboss、Spring AOP、AOP/ST 等。

静态编织中由于方面代码在程序运行前已被内联至核心功能代码中，代码被高度优化，因此，执行速度与未使用 AOP 方式编写的代码相差无几，对性能几乎没有影响；而动态编织由于引入了额外的抽象层，性能会有一定的降低。静态编织会改变类的源代

码，而且需要在设计时就要确定所有的方面；而动态编织则不会，它可以在运行时增加、修改和删除方面。

2．不同时刻的编织

正如可以用不同的机制编译和执行 OOP 程序那样，AOP 工具为构建和执行方面提供了不同的工具。方面编织器提供了按照方面中切入点指定的方式，自动调用通知的搭配方式，方面的编织对于性能和可伸缩性有影响，其中大部分取决于编织发生在应用程序生命周期的哪一部分。

1）编译时编织

编译时编织可以在编译前进行预处理，将两种代码自动混合，将方面中的代码自动插入到功能模块代码的合适位置处。也可在编译后，对编译后的代码进行操作。对于普通应用程序而言，在编译时进行编织操作是最为直观的做法。由于源程序中包含了应用的所有信息，因此这种方式通常支持最多种类的连接点。利用编译时编织，能够使用 AOP 系统进行细粒度的编织操作。

编译时编织最典型的框架有 AspectJ、AspectC++和 AspectWerkz 等。编译时编织会带来开销，主要是内存和时间的使用，因为它是在编译时执行大部分通知。在大型项目中，这些开销有可能是很可观的，而且可能带来一些问题，特别是在切入点发生变化时，大部分系统都需要重新编译。然而，编译时编织也意味着在运行的时候，几乎不需要为了匹配切入点做额外的工作。

2）载入时编织

载入时编织是在代码载入时，实现代码的编织。程序的主逻辑部分和方面部分可以分别进行开发和编译，而编织操作就发生在框架载入方面代码之时。Aspect Werkz、AspectJ、Spring 和 JBoss 都支持载入时编织。

3）运行时编织

运行时编织是指在运行时根据对方法的调用执行适当的方面代码以实现编织。运行时编织可能是所有编织方式中最为灵活的，程序在运行过程中可以为单个的对象指定是否需要编织特定的方面。运行时编织采用的编织技术有反射、动态代理和拦截器。

希赛教育专家提示：AOP 是一种新的开发方式，但它不是一种要取代传统编程方法的技术，而是对传统编程方法的有力补充。实现机制不同的 AOP 适用于不同要求的应用，从调用通知的速度来看，编译时编织的调用最快，起止是载入时编织，最后是运行时编织；而从编译时的开销来看，编译时编织的开销最多，其次是载入时编织，运行时编织几乎没有编译开销。所以，针对具体的应用场合，需要作出不同的选择。

25.2.4　横切技术

“横切”是 AOP 的专有名词，它是一种蕴含强大力量的相对简单的设计和编程技术，尤其是用于建立松散耦合的、可扩展的企业系统时。横切技术可以使得 AOP 在一个给

定的编程模型中穿越既定的职责部分的操作。

在传统的程序中，由于横切行为的实现是分散的，开发人员很难对这些行为进行逻辑上的实现或更改。例如，用于日志记录的代码和主要用于其他职责的代码缠绕在一起。根据所解决的问题的复杂程度和作用域的不同，所引起的混乱可大可小。更改一个应用程序的日志记录策略可能涉及数百次编辑。这样，即使可行，也是个令人头疼的任务。

在 AOP 中，将这些具有公共逻辑的，与其他模块的核心逻辑纠缠在一起的行为称为“横切关注点（Crosscutting Concerns）”，因为它跨越了给定编程模型中的典型职责界限。

一个关注点就是一个特定的目的，一块用户感兴趣的区域，一段用户需要的逻辑行为。从技术的角度来说，一个典型的软件系统包含一些核心的关注点和系统级的关注点。举个例子来说，一个信用卡处理系统的核心关注点是借贷/存入处理，而系统级的关注点则是日志、事务完整性、授权、安全及性能问题等。许多关注点会在多个模块中出现，如果使用现有的编程方法，横切关注点会跨越多个模块，结果是使系统难以设计、理解、实现和演化。AOP 能够比上述方法更好地分离系统关注点，从而提供模块化的横切关注点。

通过利用 AOP 技术，改变了整个系统的设计方式。在分析系统需求之初，利用 AOP 的思想，分离出核心关注点和横切关注点。在实现了诸如日志、事务管理、权限控制等横切关注点的通用逻辑后，开发人员就可以专注于核心关注点，将精力投入到解决企业的商业逻辑上来。同时，这些封装好了的横切关注点提供的功能，可以最大限度地复用于商业逻辑的各个部分，既不需要开发人员写特殊的编码，也不会因为修改横切关注点的功能而影响具体的业务功能。

AOP 应用到的横切技术，通常分为两种类型，分别是动态横切和静态横切。

动态横切是通过切入点和连接点在一个方面中创建行为的过程，连接点可以在执行时横向地应用于现有对象。动态横切通常用于帮助向对象层次中的各种方法添加日志记录或身份认证。在很多应用场景中，动态横切技术基本上代表了 AOP，其核心主要包括连接点、切入点、通知和方面。

静态横切和动态横切的区别在于，它不修改一个给定对象的执行行为。相反，它允许通过引入附加的方法字段和属性来修改对象的结构。此外，静态横切可以把扩展和实现附加到对象的基本结构中。在 AOP 实现中，通常将静态横切称为 introduce 或 mixin。

静态横切在 AOP 技术中，受到的关注相对较少。事实上，这一技术蕴含的潜力是巨大的。使用静态横切，架构师和设计师能用一种真正面向对象的方法有效地建立复杂系统的模型。静态横切允许开发人员不用创建很深的层次结构，以一种本质上更优雅、更逼真于现实结构的方式，插入跨越整个系统的公共行为。尤其是当开发应用系统时，如果需要在不修改原有代码的前提下，引入第三方产品和 API 库，则静态横切技术将发挥巨大的作用。

25.3　支持 AOP 的开发工具

目前，支持 AOP 的开发工具主要有 AspectJ、AspectWerkz、JBoss 和 Spring 等，本节简单介绍这些工具。

25.3.1　AspectJ

AspectJ 是目前最完善的 AOP 语言，由 AOP 的首倡者 Gregor kiczales 领导的一个小组提出并得到发展。AspectJ 是对 Java 编程语言的扩展，通过增加了一些新的构造块支持对横切关注点的模块化封装，通过对源代码级别的代码混合实现织入，是一种典型的使用静态织入的 AOP 实现机制。

AspectJ 为了实现 AOP 技术，引入了 4 个新的概念：连接点、切入点、通知和方面。连接点是明确定义的程序执行过程中的一个点，切入点则是指一组相关的连接点，通知定义了在连接点执行的额外实现，方面则是指对横切关注点的模块化封装实现的单元，类似于 OOP 中的类，由切入点、通知与普通的 Java 成员声明组成。

如前所述，连接点是程序执行中明确定义的点，例如，方法调用、属性调用都是连接点的例子，在连接点处可以执行预定义的额外实现。而要指明在哪些连接点上执行，则需要定义切入点，切入点可以在程序运行时匹配特定的连接点，AspectJ 中预定义了一系列标准切入点，包括方法与构造器的调用、接受调用、执行、域的访问、异常处理、实例类型匹配、类型的初始化与静态初始化等，通过这些预定义切入点的组合可以实现自定义的、复杂的切入点。在编译时，方面中的通知将被转化为标准的方法，类代码中匹配切入点的连接点将被转化为一个静态的标记点，然后，这些静态的标记点将被对通知所转化成的方法的调用所取代，由此完成两种代码的织入，最后对织入完成的代码编译为字节码，即完成了整个编译过程。

希赛教育专家提示：AspectJ 即支持编译前的预处理方式实现代码的编织，也支持编译后的字节码操作。

25.3.2　AspectWerkz

AspectWerkz 是一个动态的 AOP 框架，利用对字节码的修改实现方面的织入，并使用 Java 虚拟机的动态替换字节码的能力实现动态 AOP 的要求。

AspectWerkz 没有扩展 Java 语言，方面、通知、切入点等均使用标准的 Java 构造块（即类和方法）来实现，并使用 XML 文件定义这些构造块，此外 AspectWerkz 还支持使用 JavaDoc 标记实现的运行期属性定义。AspectWerkz 采用了与 AspectJ 相似的连接点模型，只是支持的连接点种类少于 AspectJ，通知的类型与 AspectJ 一致。

AspectWerkz 通过引入方面容器（Aspect Container）来实现对字节码的转化（即代

码织入实现动态 AOP 的要求）。方面容器管理部署好的类、方面代码，并根据 XML 文件或 JavaDoc 注释中定义的方面、通知、切入点等得到连接点处相关的方面信息，并在程序的执行中控制执行流程，在匹配的连接点处执行适当的通知。

AspectWerkz 通过类载入层次的适当位置拦截类载入，从而实现字节码的修饰。AspectWerkz 提供了两种织入模式实现 AOP，即静态织入和动态织入。

静态织入只在类载入时对字节码作一次性的转化，通过将类的方法实现移入 AspectWerkz 命名的方法中，将原方法中的代码改写，由方面容器调用适当的通知，并调用 AspectWerkz 添加的方法从而完成代码的织入。导入则由混合类型实现，用于为类增加新的方法，混合类型是一种使用接口与实现类的方式模拟多重继承的机制。AspectWerkz 通过修改字节码使被导入的类实现混合类型的接口，并在接口定义的方法中，将控制交给容器管理器，由它来完成对实现的调用。

静态织入可以在运行时动态地为切入点增加、删除参考，可以引入新的参考，但是无法定义新的切入点，这需要动态织入。动态织入由两阶段织入完成，分别为类载入阶段与激活阶段。首先，在类载入时，按照静态织入的方法，为需要实现动态织入的类的每个方法添加一个相应的空的方法，匹配连接点的方法除外。然后，在激活阶段，原方法体中的代码将被交换到类载入时新产生的方法中，原方法将实现静态织入时相同的处理，从而方面容器控制流程。前述代码交换是由热交换（HotSwap）实现的，这是 JVM 提供的 API。通过方面容器与织入模型，AspectWerkz 提供了动态 AOP 的实现。

25.3.3 JBoss AOP

JBoss 是一个开源的符合 J2EE 规范的应用服务器，作为 J2EE 规范的补充，JBoss 中引入了 AOP 框架，为普通 Java 类提供了 J2EE 服务，而无需遵循 EJB 规范。JBoss 通过类载入时对字节码操作实现动态 AOP 框架。在 JBoss 中切入点与方面也由普通 Java 对象实现，并使用 XML 文件配置。JBoss 的连接点模型与 AspectJ 略有不同，提供了一系列预定义的切入点，包括类匹配、方法调用、构造器调用、域访问、特定的调用与被调用关系。通过这些切入点的逻辑运算，可以实现更为复杂的切入点。方面为 Java 类，通知是其中的一个方法，方面中不含切入点，方面主要是各种拦截器（只含一个通知的方面），单一连接点上可由多个拦截器形成拦截器链，拦截器执行额外的操作。此外，JBoss 提供对元数据的支持，用于为类、方法、构造器以及域添加额外的属性，并可在运行期访问。

为实现拦截，JBoss 需要修改类的字节码，大致过程如下：

XML 配置文件中关于切入点、拦截器、元数据以及混合类的信息在应用程序部署时被读入、解析，并生成相应的对象，这些信息与实例化的对象由 AspectManager 管理。在需要混入方面代码的类载入时，AspectManager 将创建 Advisor 类，将方面相关信息传递给它，并对类的字节码进行修改。然后，将修改过的字节码交给类载入器完成类的装

载。字节码的修改主要是对被载入的类添加一系列方法用于代理那些匹配连接点的方法调用、构造器调用、域访问，以及方法导入，转为对 Advisor 类相应方法的调用。类中各方法将重命名，保留原方法体，并添加一个与原方法同名的方法，在这个方法中调用那些代理方法，用来将调用代理给 Adviosr 类，或调用重命名的原方法。

对于域访问，分别添加两个方法，对应于读写操作，将域访问代理至 Advisor 类，在访问这个域的类中，则需将对域的访问转换为对上述方法的调用；对于构造器调用，则添加一个方法，将调用代理至 Advisor 类，并对构造对象的类构造代码作相应转换；对于导入，被导入的类中将添加一个混合类实现的引用，并添加混合类接口中的方法，将对混合类方法的调用代理至 Advisor 类，并最终调用混合类的实现。相关类载入后，初始化 Advisor 类，填入拦截器链，以完成整个处理过程。

25.3.4　Spring AOP

Spring 是一个基于反转控制（Inversion of Control，IoC）的轻量级 J2EE 框架，它的核心是 IoC 容器。IoC 是一个基于构件的设计模式，它不是由构件本身来判断彼此间的依赖关系，而是将判断依赖关系的职责移交给容器，这大大提高了构件的可移植性和可重用性。Spring 采用 Java 的反射机制，动态调用以避免硬编码方式的约束，并在此基础上，建立了其核心构件 BeanFactory 模式，以此作为其依赖注入机制的实现基础。

Spring AOP 默认使用 JDK 提供的动态代理机制，业务对象通过接口编程。它对于有接口的类使用的是 Java 内部类提供的代理，而对于那些不实现接口的类使用的是开源的 Cglib 库，动态创建一个子类来实现。

Spring 的 AOP 实现不同于 AspectJ、AspectwerkZ、JBoss 等，它不是完全的 AOP 实现，而是设计用例与 IoC 容器配合使用，在应用服务器环境下实现 AOP，目前不支持字段拦截，只支持方法拦截。由于 Spring 的核心是 IoC，因此，方面、连接点和切入点等，都是通过配置 XML 文件实现。

25.4　AOP 的应用

AOP 可应用的范围非常广泛，作为一种思想，跨越多个语言平台，用于各种各样的实际应用。以下是部分已经经过实践检验的应用场景：

（1）应用系统中的日志功能、权限认证、事务处理、缓存管理等关注。

（2）实时操作系统中将实时关注作为一个方面提取出来，形成一个独立于系统的时间方面，实现时间方面的并发设计和系统时间特性的统一管理。

（3）对工作流系统进行改造，将影响所有业务模块的流程管理功能分离出来，形成一个流程方面，各个业务模块都会被这个方面“监视”。

（4）在 C++程序开发中，将内存管理作为一个关注点，形成内存管理方面，用于帮助检查内存泄漏及其他内存管理；改进已有的各种面向对象的设计模式，实现关注点的分离，使模式具有更好的模块化结构。

与其他的编程思想发展过程一样，面向方面作为一个后来者，其理论研究与实践应用的事件都还不长，仍然有许多问题需要进一步的研究与探讨。主要集中在以下几个方面：

（1）如何有效地判断和识别一个系统中的横切关注点。

（2）如何准确把握方面之间及其与核心类之间的相互关系。

（3）如何进一步开发和丰富支持 AOP 的语言并保证其正确性。

（4）如何利用现有的 CASE 工具为 AOP 系统建模。

（5）如何开发有效的工具来满足从软件设计到维护各个阶段的需要。

AOP 的出现是为了解决一些传统方法中不能很好解决的横切关注的问题，它不是一种取代传统编程方法的方法，而是对于这些方法的补充，它将软件系统中不同维度的关注点分离，避免将横切方面的关注分散在核心的业务代码中。

目前，AOP 仍然处在迅速发展的阶段，但已经出现在众多的实际应用与学术研究中，特别是一些流行的应用开发框架已经集成了对于 AOP 的支持，在开发人员中广为流传。但是，在软件系统前期的需求分析及设计阶段，并没有很好的方法对 AOP 进行支持。随着 AOP 的思想更深地融入软件工程，进一步影响到软件的前期分析与设计阶段，可以预见，它给软件开发方法学带来的是全方位的变革，而不是仅仅停留在实现层面。

本章参考文献

[1] 莫倩，王恺，刘冬梅. 面向方面的软件开发. 北京：机械工业出版社，2006

[2] 梁海华. 产生式编程——方法、工具与应用. 北京：中国电力出版社，2004

[3] 范国强. 面向方面开发方法的一种改进. 同济大学硕士学位论文，2006

[4] 关少珊. 基于扩展 UML 的面向方面建模的研究. 汕头大学硕士论文，2006

[5] 杨军. 面向 Aspect 编程的研究与应用. 湖北工业大学硕士论文，2005

[6] 幸俊. AOP 开发过程研见及应用. 成都理工大学硕士论文，2007

[7] 曾路，张立臣. Weave.Net——基于.Net 平台的 AOP 技术. 计算机应用研究，2005（5）：225-226

[8] 曹东刚，梅宏. 面向 Aspect 程序设计——一种新的编程范型.计算机科学，2003（9）：1565-1572

[9] 何丽莉. 关注分离问题研究综述. 计算机科学，2005(2):129-131

[10] 潘哗，任广伟. 基于 AoSD 的 web 架构规划. 计算机技术与发展，2006（12）：

52-54

[11] 朱志军，庄成三，郑炳伦. 基于.Net 的轻量级 AOP 技术.现代计算机，2005（3）：19-22

[12] 郭东亮，张立臣. 基于扩展 UML 的面向方面的建模. 计算机工程，2006（10）：100-102

[13] 李志纯，张南平. 面向 Aspect 编程的应用研究. 计算机技术与发展，2006（5）：217-222